(2)

RL-TR-91-231
Final Technical Report
September 1991

ACTIVE DATA/KNOWLEDGE BASE DICTIONARY

Dove Electronics, Inc.

J. Dove, L. Kerschberg, P. Berra, C. Bosch, D. Weishar,
A. Waisanen, J.P. Yoon

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-91-231 has been reviewed and is approved for publication.

APPROVED: Robert M. Flo

ROBERT M. FLO
Project Engineer

FOR THE COMMANDER: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Technical Director
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL(C3AB) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1991	Aug 89 - Apr 91

4. TITLE AND SUBTITLE
ACTIVE DATA/KNOWLEDGE BASE DICTIONARY

5. FUNDING NUMBERS
C - F30602-89-C-0068
PE - 62702F
PR - 5581
TA - 21
WU - 92

6. AUTHOR(S)
J. Dove, L. Kerschberg, P. Berra, C. Bosch, D. Weishar, A. Waisanen, J.P. Yoon

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Dove Electronics, Inc.
227 Liberty Plaza
Rome NY 13440

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3AB)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-91-231

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Robert M. Flo/C3AB/(315) 330-2805

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This effort was to explore data and knowledge-based processing systems with particular attention to:

Adequacy of the Information Resource Dictionary Systems (IRDS) to support advanced systems development projects;

Conceptual architecture for an Active Data/Knowledge Dictionary System;

Exploring in detail intelligent query formulation and processing in heterogeneous database systems; coordinated problem solving with multiple heterogeneous knowledge sources; schema evolution in object-oriented databases; Hypermedia requirements for active dictionaries, and providing a better understanding of the role of metadata in the management of knowledge-intensive object-oriented applications.

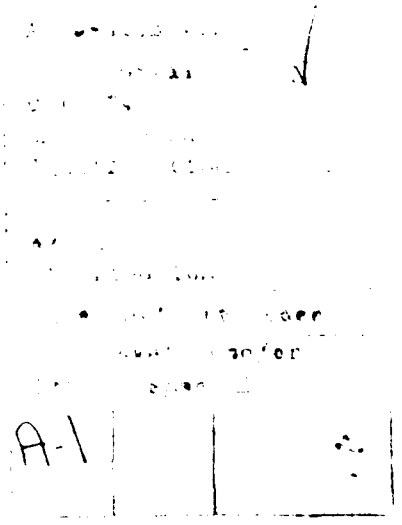
14. SUBJECT TERMS
Data Dictionary, Database Management, Knowledge Data Language

15. NUMBER OF PAGES
138

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

Table of Contents



List of Tables

List of Figures

1 Introduction

1.1 Motivation

The Active Data/Knowledge Dictionary project is motivated by the need to support the entire life-cycle for developing data and knowledge-based applications; this includes the traditional phases of requirements analysis, requirements specification, database and knowledge base design, application program design, implementation, testing, maintenance and evolution.

Our view is that database systems and knowledge based systems will grow closer together, especially as new paradigms are developed to find useful interactions between knowledge bases and databases. Some of these paradigms have already begun to appear. For example, many applications have been reported where an expert system is loosely-coupled with a database system. The database is used as a data server so that an expert system can ask for data to continue its reasoning and problem-solving tasks.

More importantly, considerable research is underway to integrate certain knowledge representation and knowledge processing schemes directly into the database. A notable example is that of *active databases* where Artificial Intelligence production systems are integrated directly into the database. In this way a production system can be used to monitor the database state and to execute situation-action rules against a relational database; the firing of these rules effects changes to the database state which can trigger new rule firings. The rules that are executed can alert application programs, update the database, or send requests to more powerful inference mechanisms such as expert systems or truth maintenance systems.

Another important trend is that of object-oriented database systems. Although these systems are in their infancy, it is important understand how object-oriented systems can be used to support new types of applications. An important research area is that of merging object-oriented concepts with the more traditional approaches of semantic data modeling, logic programming and knowledge representation techniques.

Not only is new database and knowledge based technology being developed, but also new applications are being envisioned in which the data and knowledge relationships are very closely associated and their logical and physical structures are very complex. For example, in the area of Software Engineering there is a need to provide support for the entire systems development life-cycle from requirements, to specifications, to design, to code and maintenance. The Computer Assisted Software Engineering (CASE) tools are intended to support several phases of the life-cycle, and it is clear that object-oriented techniques are needed to manage the large repositories of heterogeneous knowledge and data representations (objects) that comprise a software system.

Similarly, the development of knowledge based applications requires that large collections of rules or some comparable knowledge representation be evolved incrementally. The management of consistency among these representations and the validation of the knowledge are important to the development, quality and reliability of these systems.

Finally, in database applications we find several trends which will require advanced support environments. Many organizations are finding that databases developed over several years by separate organizations contain overlapping information; they are forming federations of possibly heterogeneous database systems. These enterprise-wide applications must have access to a

coherent enterprise information architecture. One important issue is how the *structure* and *meaning* of each database should be represented to the members of the federation. Another is how queries involving multiple databases can be processed efficiently.

Another important application involves database design in the context of object-oriented data models and systems. The issue here is that of *schema evolution*, given that objects instances and object types can be created and changed dynamically. This is a revolutionary concept in the light of traditional database systems in which the database schema is fixed and very difficult to change.

1.2 Goals of the Active Data/Knowledge Dictionary project

The Active Data/Knowledge Dictionary (ADKD) project is intended to explore several of the above mentioned areas. The major goals are:

1) To explore the adequacy of the Information Resource Dictionary Systems (IRDS) to support advanced systems development projects,

2) To propose a conceptual architecture for an Active Data/Knowledge Dictionary System based on the requirements of the above-mentioned advanced application areas,

3) To explore in detail certain areas of active data/knowledge management, with particular attention to:

   - Intelligent query formulation and processing in heterogeneous database systems,
   - Coordinated problem solving with multiple heterogeneous knowledge sources,
   - Schema evolution in object-oriented databases,
   - Active data/knowledge management in object-oriented databases,
   - Hypermedia requirements for active dictionaries, and

4) To provide a better understanding of the role of meta-data in the management of knowledge-intensive object-oriented applications.

1.3 Organization of the report

Section 2 provides an overview of data dictionaries with particular emphasis on the Information Resource Dictionary System which is designed to support and manage software applications. We provide a critique of the IRDS and discuss both its strengths and weaknesses. Section 3 explores the conceptual architecture for an Active Data/Knowledge Dictionary and suggests that the dictionary should provide a set of tools to assist the user in developing data/knowledge applications. The report continues with several sections dealing with specialized research areas in which the active data/knowledge dictionary concept is of great importance.

Section 4 discusses intelligent query processing in heterogeneous databases. A survey of system prototypes is presented and the techniques used for query processing are discussed. Next the Knowledge/Data Model is introduced and a novel architecture, the Intelligent Heterogeneous Autonomous Database (InHead) architecture is presented. Two examples show how the InHead intelligent thesaurus and blackboard architecture are used for cooperative query formulation and processing. Section 5 focuses on the problem of using multiple heterogeneous knowledge

representations to solve problems associated with 1) telecommunication network fault diagnosis and 2) database query optimization. The major knowledge representation schemes that are addressed are model-based, case-based and rule-based viewpoints of the objects in the system. The role of the intelligent thesaurus is discussed in this context.

Section 6 examines object-oriented database systems and the problem of schema evolution in object-oriented databases. The active dictionary makes use of an assumption-based truth maintenance system to maintain and control the various versions of classes and schemata defined during the database design process. An example is provided for the schema evolution problem.

Section 7 examines knowledge/data management in active object-oriented databases. Here, production rules are integrated into an object-oriented data model. A novel concept, the Rule Schema, is used to reason about the dependencies among rules, objects and constraints in an object-oriented database schema. Hypermedia requirements for active dictionaries are discussed in section 8. Section 9 presents our conclusions and suggestions for future research. Section 10 is our bibliography and it is followed by two appendices. Appendix A describes the concepts of blackboard systems and appendix B gives a sample session with the GemStone System.

2 Data Dictionaries and Directories

Data dictionary and directory systems are an integral part of data administration environments which manage the information resources of the organization. Data dictionaries may be used to catalog information regarding both automated and non-automated enterprise resources; these may include paper records, database files, schemata and catalogs, as well as information regarding hardware, software and human resources.

The data dictionary is the repository of all definitions, that is meta-data, associated with the enterprise. Usually we connote the dictionary as being associated with a Database Management System (DBMS), but at times it is convenient to extend the scope of the dictionary/directory to include the information resources of the entire enterprise.

In this section we focus on the role of data dictionaries in support of database applications. Section 2.1 provides an overview of the major functions of the data dictionary/directory and defines both passive and active dictionaries. Section 2.2 presents the architecture and a discussion of the Information Resource Dictionary System (IRDS) which is an ANSI standard for such systems.

2.1 Definition and functions of the data dictionary/directory

In an excellent survey of data dictionaries and directories (D/D), Allen, Loomis and Mannino [ALM82] define the D/D as follows:

Definition 2.1 A D/D system is an automated information system composed of:

- a *database*, called the D/D, that contains the meta-data describing the data, processes, users, and processors of an organization;
- *retrieval and analysis capabilities* that assist a wide range of user groups in application development;
- *management tools* that help ensure the security, validity, recoverability, integrity, and shared accessibility of the D/D;
- *functional interfaces* that permit other software modules to access the D/D and convert meta-data into the format required by the D/D System.

The main objectives of the D/D are to allow data processing personnel to control the data item definitions used by all people and programs in the organization, to control the cost of developing, maintaining and evolving applications, to provide management reports on database and dictionary usage patterns, and to define, store and manage corporate meta-data within a single repository.

Dictionary Users

Allen et. al. [ALM82], identify six user categories for a D/D system. They include:

1) Data administrators — use the D/D to design, monitor and restructure databases. They also use the D/D to maintain a common repository of data element definitions and implement standards for their use in application development,

2) Application personnel — use the D/D to access, store and manage programs, schemata and configurations for system evolution,

3) Operations staff — use the D/D to monitor the performance of the system for better job scheduling,

4) Data processing management — use reports from the D/D to determine data usage,

5) End users — have access to their views and the database schema information by means of the dictionary, and

6) Auditors — monitor and examine system definitions, documentation and performance with the assistance of the D/D.

Data Dictionary Organization

The D/D provides a well-defined logical structure, or schema, by which the enterprise objects and concepts can be described. For example, the data dictionary may describe the enterprise as consisting of the following types of entities: Users, Databases, Sub-schemata, Relations, Attributes, Domains, Tuples, Files, Transactions, Processes, Processors, Communication Lines, and Workstations. The D/D also provides mechanisms for these concepts to be related. For example, in the IRDS to be discussed in section 2.2, the Entity/Relationship Model [Che76] is used to represent the Basic Functional Schema provided by the IRDS. Users may extend the schema by defining additional entity sets and relationship sets.

Clearly, the predefined D/D schemas are intended to assist in the management of data processing organizations that develop applications involving the access and manipulation of large shared databases. These modeling techniques could be used to provide schemata for other development environments, as for example, software engineering environments or knowledge-based systems development. These will be addressed in section 3.

Data Dictionary Functions

The D/D has several major functions that it provides to its user community. These include the following:

- *D/D Maintenance* function provides update capabilities so that D/D elements can be defined, modified, and deleted from the dictionary database.
- *Extensibility* function allows the predefined D/D schema to be extended with new structures, that is, new entity sets, relationship sets and their attributes.
- *Report Processor* function provides general reporting capabilities as well as user-defined reports.
- *Query Processor* function allows users to query the dictionary interactively in much the same way a user would query the actual database.
- *Convert Function* of a D/D system is capable of reading application programs, libraries, and schemata and generate for the D/D Maintenance Function a set of input transactions that can be used to initially load the D/D with meta-data from various applications that will now be managed by the D/D.

- *Software Interface* function allows the D/D to format its meta-data for other programs such as compilers and data definition processors to use. This function thus serves to provide multiple representations of the meta-data to those tools associated with the management of database applications.
- *Exit Facility* function allows the D/D procedures and programs to be extended by local users to reflect the procedures of the organization. For example, the local organization might wish to implement a different access authorization policy that would required a locally developed program.
- *D/D Management* function performs the same database management tasks associated with the database itself. These include security, concurrency control, integrity management, etc.

The next section studies one such dictionary system, the Information Resource Dictionary System, in detail. We show that the IRDS is structured to support multiple levels of meta-data. We argue that the IRDS, while suitable for many traditional data processing applications, will fall short in terms of functionality for the new classes of applications involving both object-oriented and knowledge-based concepts.

2.2 The Information Resource Dictionary System

The Information Resource Dictionary System (IRDS) has been developed by Technical Committee H4 of the Accredited Standards Committee X3 (X3H4) of the American National Standards Institute (ANSI) in conjunction with the National Computer and Telecommunications Laboratory (NCTL) at the National Institute of Standards and Technology (NIST). The purpose of the IRDS standard is "to provide U.S. Federal Government data dictionary system users with useful, flexible, and user-friendly data dictionary system software products to support all phases of the system life cycle."

The IRDS standard was developed with the following design objectives:

- The IRDS should contain the major features and capabilities that exist in currently available dictionaries,
- The IRDS should be modular to support a wide range of user environments and to support cost-effective procurement, and
- The IRDS should support portability of skills and data.

In following the first of these design objectives, U.S. Federal Government representatives and dictionary software vendors reviewed draft versions of the IRDS Specifications. This resulted in an IRDS Specification containing the most commonly used facilities of existing systems, and, as of 1988, an IRDS Specification representing the "state-of-practice" in data dictionary systems technology.

To provide IRDS flexibility and procurement cost-effectiveness, X3H4 and NIST adopted a modular approach. The ANSI X3H4 IRDS Standard consists of specifications for a Core Standard data dictionary system module and for five additional modules: the Basic Functional Schema, IRDS Security, the Extensible Life Cycle Phase Facility, the Procedure Facility, and the Application Program Interface. The additional modules are optional. Thus, IRDS users need to

procure them only if their application requires that functionality. To provide additional flexibility, capabilities are specified in the Core IRDS that enable users to customize or extend the type of data that can be stored in the Information Resource Dictionary (IRD).

The result of the third design objective is that the IRDS Specifications include a menu-driven panel interface for inexperienced users and a command language interface for more experienced users.

2.2.1 The major IRDS functions

Levels of data and meta-data in the IRDS

The IRDS architecture is specified in terms of the Entity-Relationship-Attribute (ERA) model and is comprised of: 1) data in a production database, 2) an Information Resource Dictionary (IRD) and 3) an accompanying IRD schema. In the Entity-Relationship-Attribute (ERA) model, entities represent "real-world" concepts such as persons, events, or quantities. These entities are represented in collections called Entity Sets. Relationship Sets are used to describe the associations between entity sets, while attributes represent properties of both entity sets and relationship sets. Instances of entity sets, relationship sets and attributes comprise the data in the production database.

The Information Resource Dictionary consists of entities, attributes, and relationships that are instances of the corresponding IRD schema entity-types, relationship-types, and attribute-types. The IRD schema, in turn, consists of instances of meta-entities, meta-relationships, and meta-attributes at the IRD schema description level. These various meta-data levels are summarized in Table 2.1 and a description of the major IRDS Modules follows.

The levels in Table 2.1 range from level 0 to level 4. The IRDS contains levels 1 through 4, and we have added a Level 0 which corresponds to the primitives to be used in constructing the higher-numbered levels. Level 0 would use an object-oriented data model whose primitives could be used to implement the ERA Database Model for level 1; the level 1 ERA Data Model would be used to specify the level 2 IRD Schema Constructs; the level 2 IRD Schema Constructs in turn would be used to specify an ERA Application Schema; and level 4 would manage the production database organized according to the level 3 schema.

The major IRDS functions and facilities are presented in more detail.

The Core Standard Module

The IRDS Core Standard Module provides the foundation for all IRDS schema structures. Within the Core Standard Module, meta-entities, meta-relationships and meta-attribute types are defined from which all other meta-levels are constructed. Among the meta-entities defined in the IRDS Core are a number of meta-attribute types that can be used for IRD meta-data control and partial validation.

The Core Standard Module contains the Minimal Schema, which the IRDS Standard specifies for every IRDS implementation. The Minimal Schema provides the critical schema descriptors needed to control every IRD schema and IRD.

Table 2.1 Levels of Data and Meta-Data in the IRDS

Level	Meta-Level	Concepts	Information Type
Level 0	Meta-Meta-Meta Schema	Object-Oriented Data Model Modeling Primitives	Objects Types, Methods, Inheritance, Constraints, Rules, Knowledge, etc.
Level 1	Meta-Meta-Schema	ERA Data Model Implementation using primitives	Core Standard Module Entity-Types, Relationship-Types, Attribute-Types
Level 2	Meta-Schema	Pre-defined IRD Entity, Attribute Relationship Schema Types using ERA Data Model	Basic Functional Schema IRD Schema Constructs
Level 3	Application Schema	Application Schema using IRD Schema Types	Customer Entity Set, Invoice Entity Set, Billing Relationship Set
Level 4	Real-World Business Concepts	Instances organized according to Application Schema	Customers are billed through invoices

The Basic Functional Schema

The Basic Functional Schema provides an initial set of schema structures. This set reflects agreements reached by members of X3H4 and attendees at user workshops sponsored by the National Bureau of Standards. These groups believed that the entity-types, relationship-types and attribute-types specified in the Basic Functional Schema could describe most existing and planned manual and automated systems.

The IRDS Security Module

The IRDS Security Module provides for two levels of access control: global security, which allows IRDS users to specify views over sets of entities, relationships and attributes; and entity-level security, which allows users to assign read and/or write privileges to specific entities.

The Extensible Life-Cycle-Phase Facility

The Extensible Life-Cycle-Phase Facility specifies integrity rules and customization facilities that give IRDS users life-cycle support. This facility allows specific entity occurrences to be associated with software life-cycle phases, thus providing control for the movement of entities through the life-cycle (i.e., a measure of version control). The Extensible Life-Cycle-Phase Facility has capabilities for including Hierarchical Phase Modeling, Relationship Sensitivity Structures, and Life Cycle Integrity Rules.

The IRDS Procedure Facility

The IRDS Procedure Facility provides the ability to define, store, maintain and execute procedures involving the IRD and the IRD Schema. These procedures are composed of IRDS Commands, along with flow-control and assignment statements.

The IRDS Application Program Interface

The IRDS Application Program Interface provides an interface between standard programming languages and the IRDS command language. With this module users can write programs to retrieve meta-data from, and pass meta-data to, the IRD.

2.2.2 Advantages and disadvantages of the IRDS

There are several advantages to using the IRDS Standard. Firstly, it is an advantage in and of the fact that it is a standard. The standard provides a common framework, terminology, and methodology for data element standardization throughout the software life-cycle. The IRDS Standard also provides system designers with the expressive power of the Entity-Relationship-Attribute data model. And finally, from a Federal Government perspective, there is a cost savings advantage in that NIST has implemented a prototype which is freely available to Federal Agencies.

On the other hand, there are several problems with using the IRDS Standard. Firstly, is its confusing terminology. The system description is replete with meta-, meta-meta-, and meta-meta-meta-entities, relationships and attributes. This makes it very easy to get lost in the meta-levels. Even the definition of one of its fundamental parts, the attribute, causes confusion. As mentioned earlier, attributes represent properties of entities or relationships. On the surface, the definition coincides with normal database terminology. But deeper investigation reveals that an IRDS attribute is really describing properties relating to entity types or relationship types rather than describing properties of entities or relationships themselves. For example, one might think of an employee record with fields for name and social security number. In generally accepted database terminology, the employee record would be the entity, with attributes name and social security number. However, in accordance with the IRDS Standard, employee record, name and social security number would all be entities. Attributes for these entities would be descriptors such as field length, or type string.

Another drawback in the IRDS Standard is that it allows only binary relationships between entities. Other than binary relationships are a common occurrence is everyday life. The family relationship is a good example. Entire classes of entities (parents, grandparents, aunts, uncles, etc.) could participate in a single relationship. There is no direct way of representing this in the IRDS Standard.

The IRDS Standard is also problematic in that attributes cannot be entities. Consider for example personnel and payroll systems that have a number of entity types and entity instances in common. For consistency and interoperability, these common entities should be modeled as attributes (as entities) of the two systems' interrelationship. Since attributes cannot act as entities, one must take the relationship and decompose it into two additional relationships, where the entities actually exist within the relationship definition.

Each entity-type name and entity name must be unique throughout an IRD. In certain applications, such as representing the information within a federation of heterogeneous databases, this is a

severe constraint. In effect, this dictates that one must define an integrated global schema. In a global schema, identically-named entities and entity-types have the same meaning unless the global data model has the capability to distinguish semantic variations. The ERA model does not have this capability.

Other characteristics of the IRDS Standard make it cumbersome. The IRDS developers state that the Standard is easily extended. In reality, extensions for all but simple objects and relationships are complex. All of the various combinations of associations between entity-types, attribute-types and relationship-types must be supplied by users.

Finally, the IRDS Standard is not object-oriented. There is no inheritance, nor is there the capability to use methods and pass messages between entities. Also, there is no capability to represent heuristics, constraints and temporal relationships. One is left with a one-dimensional structural representation, with the dynamics and the constraints of the application omitted. These would have to be implemented in the application programs accessing the dictionary and the actual database. In subsequent sections we propose an object-oriented characterization of an active data/knowledge dictionary that overcomes many of the above-mentioned limitations of the IRDS.

3 Active Data/Knowledge Dictionary System

The data dictionary/directory technologies [ALM82] developed for the 1980's must be evolved and extended to handle new requirements imposed both by new database system architectures and by the new applications that are being planned, designed and implemented.

Since 1984 much attention has been focussed on the field of Expert Database Systems which studies the models, tools, techniques and architectures for integrating Artificial Intelligence (AI) and Database Management (DB) systems to provide knowledge-directed reasoning over large, shared databases. In addition, object-oriented technologies from the fields of knowledge representation, databases, and programming languages are maturing and are being incorporated into these new object-oriented database systems.

Application developers and system architects are creating more sophisticated applications involving both knowledge-based and data-based components. In many cases these architectures involve existing systems that must now communicate and cooperate to achieve the strategic information goals of the organization. Thus information may be distributed among several possibly overlapping and redundant systems.

Our goal in this section is to explore the concepts, tools and techniques that will be needed to support the development of knowledge- and data-based applications of the 1990's. Clearly, data dictionaries will require enhanced capabilities to support these applications as well as the entire systems life-cycle. In this section we provide a definition, conceptual architecture and functional specification for an Active Data/Knowledge Dictionary (ADKD) *environment* for the development of expert database applications. In subsequent sections of this report, we present the results of our research into some underlying tools and techniques for the ADKD.

3.1 Definition of an active database and active dictionary

Definition 3.1 A database management system is said to be *active* if it can monitor the database state and automatically perform operations that 1) alert application programs, expert systems, or other knowledge-based systems, and 2) update the database through rule-generated updates.

A popular active database architecture is one that integrates an AI production system with a relational database system as depicted in figure 3.1.

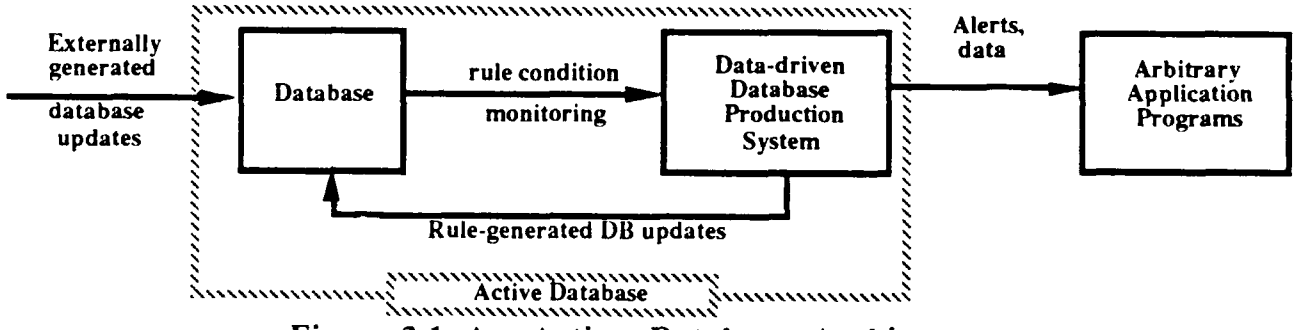


Figure 3.1 An Active Database Architecture

Definition 3.2 An data dictionary/directory system, as defined in Definition 2.1, is said to be *active* if it can monitor the state and performance of the underlying databases it supports and be both proactive and reactive in suggesting and effecting needed evolutionary changes in those databases.

This definition distinguishes an active dictionary from a passive one which is consulted for data definitions and descriptions, but does not actively participate in the enforcement of the data definitions and other database constraints.

Definition 3.3 An Active Data/Knowledge Dictionary (ADKD) is an active data dictionary/directory system which supports both data- and knowledge-based applications. The ADKD is an automated ***information environment*** composed of:

- a data and knowledge base containing the meta-data and meta-knowledge characterizing the data, knowledge, processes and rules of an organization,
- browsing, retrieval and analysis capabilities to assist a wide range of users in the development and management of large data- and knowledge-based applications,
- tools to allow users to define, maintain and evolve data/knowledge representations and applications, as well as to maintain the security, integrity, consistency, and shared accessibility of the data and knowledge bases, and
- functional interfaces that permit software tools to access the data and knowledge bases to obtain relevant data/knowledge specifications in formats suitable for their particular processing requirements.

The definition of the ADKD is quite general, but it does point out that new applications will have both a database component and a knowledge-based component, and these may actually be distributed and heterogeneous in nature. The ADKD must provide tools to define, manage and evolve these systems. It will have to support multiple knowledge representations and reasoning paradigms as well as multiple data models.

We feel that an *object-oriented* approach is required to provide the underlying data/knowledge modeling support for the ADKD. These concepts correspond to the Level 0 constructs presented in Table 2.1 of section 2.1. We would intend these modeling primitives to be general enough to model several well-known knowledge representations such as production rules, cases, and semantic nets as well as several semantic data models such as the functional data model, the entity--relationship model, and the relational model.

Further, it is imperative that the ADKD provide modeling capabilities to specify knowledge declaratively, rather than procedurally. In many object-oriented models, the behavioral aspects are specified as "methods" or "active values" and are written as procedures or functions in a procedural programming language. In order to be able to reason about these behavioral aspects of a representation, they must be specified in a declarative language. We agree with Dayal et al [DBM88] that "rules are objects too."

3.2 The ADKD environment

The conceptual architecture for the ADKD Environment is depicted in figure 3.2. Note that the architecture of the ADKD supports multiple users who interact with the Dialog Manager to determine which service or services are to be used. The Dialog Manager in turn works with the ADKD Coordinator to ascertain the appropriateness of the services requested and then works with the services and the ADKD Object Manager to access the required objects — both knowledge and data — from the Object Repository.

The Object Repository must have the capability to define, store, retrieve, and manage objects with very different specifications in a uniform and integrated fashion. The object-oriented paradigm suggested for the ADKD is suitable for providing object storage and retrieval services for complex and persistent objects. To support information interchange, the repository will have translators to map an object specification into the respective formats required by each tool.

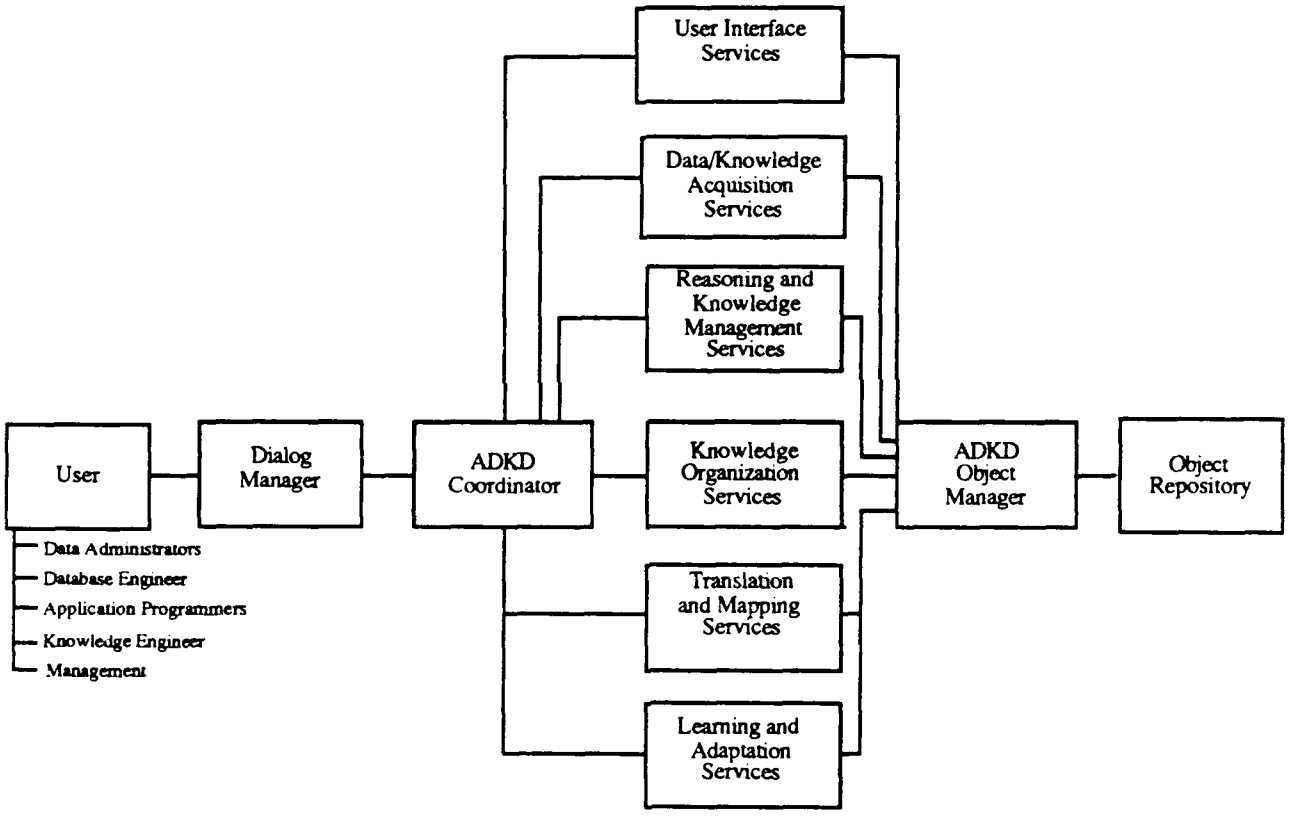


Figure 3.2 ADKD Conceptual Architecture

We now present each of the major services of the ADKD in terms of a functional architecture.

3.3 ADKD functional architecture

The Active Data/Knowledge Dictionary is envisioned as an *environment* for the development and maintenance of data- and knowledge-based applications. It consists of a set of generic tools that could support various application areas such a computer-aided Software Engineering (CASE), computer-aided design and manufacturing (CAD/CAM), as well as computer-aided database/knowledge base design (CAD/KD). The following discussion follows the diagram of

figure 3.2. The various ADKD tools and their possible interrelationships are shown in figure 3.3 below. Note that in figure 3.3 the ovals represent services and tools while the white boxes represent memories, data and knowledge structures, as well as existing enterprise systems such as the IRDS, relational DBMS and multimedia DBMS.

3.3.1 User community

The ADKD user community consists of a number of individuals who may play multiple roles. The major roles are those of Data Administrator, Database Engineer, Application Programmer, Knowledge Engineer and Management. Brief descriptions of each user follows:

Data Administrators are concerned about the overall information architecture of the enterprise, including both the automated data/knowledge systems and the non-automated ones as well.

The Database Engineer is primarily concerned with the specification, design, implementation, and management of all the database schemata of the enterprise. These schemata may be represented in diverse data models and may operate under different DBMSs. The Database Engineer should also be versed in the new object-oriented methods for database design which would be supported by the ADKD tool set.

The Application Programmer will deal with the creation of traditional database views for end users and will specify database transactions that will access the ADKD and the actual data/knowledge bases. This task of writing application programs will be assisted by the use of high-level constraints that traditionally have been embedded within application programs, but which are now within the meta-data in the ADKD; the suggested object-oriented model will capture the behavioral semantics of objects within the schema as rules and constraints.

The Knowledge Engineer is responsible for acquiring and representing, within the ADKD, corporate knowledge associated with major enterprise tasks and activities. This knowledge when matched with appropriate data elements produces information used by the enterprise. The Knowledge Engineer must work closely with Data Engineers to define appropriate data/knowledge representations and mappings to/from the data/knowledge bases to obtain translations and to tune the overall functionality and performance of the system.

Management is concerned with the overall operation and performance of the ADKD within the goals and mission of the organization. The ADKD should have the capability to provide pre-defined reports to management and to support ad-hoc queries using both a standard SQL interface as well as an object-oriented browser.

3.3.2 Dialog Management

The ADKD Dialog Manager (DM) assists the user in accessing the appropriate tools to perform tasks associated with data and knowledge management. The DM compiles user profiles for authorized users and uses them to structure the dialog and to provide authorized services. Once the intent of the user is determined, the DM interacts with the ADKD Coordinator to access the suite of tools and services to accomplish the intended task.

The Dialog Manager should allow users to use command line instructions such as those offered by DOS and Unix-based systems. It should also provide more object-oriented menu-driven interfaces with pop-up menus and nested selections.

One could also posit that the Dialog Manager could have the structure of an expert system with a psuedo-natural language interface, a knowledge base of rules regarding the use and function of the ADKD and explanation facilities to assist users in understanding the complex data/knowledge organizations of the ADKD.

3.3.3 ADKD Coordinator

The ADKD Coordinator is the "traffic-cop" that controls access to ADKD services. Here too, the coordinator has access to a knowledge base that serves as a guide to the inter-relationships among the ADKD services and how they may support a user-defined task. Consider for example that the enterprise has a set of well-defined procedures for the design of an object-oriented database schema and that specific rules exist for possible schema evolution[1]. These methods are stored in a knowledge base within the Object Repository and are used to evaluate the "state" of the database design activity; the appropriate ADKD services can then be provided to assist the user in this activity. There may be certain configurations of services that would be inappropriate because they would conflict with enterprise rules. Thus, we see the role of control knowledge, or meta-knowledge, and how it can affect the use of meta-data stored in the object repository.

3.3.4 User Interface Services

The User Interface Services are those directly related to allowing the user community to access the ADKD data and knowledge sources. These interfaces should included object-oriented browsers that present the complex data and knowledge organizations in the object repository. At a minimum, the user interface services should describe the object types, their relationships, rules associated with object types, attributes, relationships and constraints on the schema. Additional information can be provided in the form of inheritance hierarchies and lattices, and the aggregation hierarchies of composite object types.

The use of hypertext and hypermedia types would be especially useful in allowing the user to freely associate objects through hyperlinks. Also, in certain situations it is advantageous to be able to annotate a design decision by attaching a diagram, an image, or a voice annotation to a particular object type or to a subschema[2].

3.3.5 Data/Knowledge Acquisition Services

The ADKD must support the acquisition, representation, storage and maintenance of multiple data and knowledge models. By model we mean the structure, semantics, operations, constraints, and query processing and reasoning services associated with a model. In order to support existing systems, the ADKD should provide abstract models for file systems, relational database systems,

---

1 Specific approaches to schema evolution in object-oriented systems are discussed in section 6.

2 Section 8 discusses the modeling of hypermedia object types in the context of multimedia documents, the IRDS, and the ADKD.

network data models, and others. The more advance semantic models such as the functional and entity/relationship models should also be supported.

The knowledge representation schemes should include production rules, semantic networks, frame-based systems, and case-based systems. The meta-data associated with these KR schemes should be represented and stored explicitly. Another important requirement of the Knowledge/Data Acquisition services is that the evolution of the various schemata must be supported[3].

At this point, it is too early to commit to a particular object-oriented model[4],but we feel that such a model could serve as the "primitive model" in which the other paradigms could be expressed. This is a topic of ongoing research.

3.3.6 Reasoning and Knowledge Management Services

The ADKD must perform complex reasoning tasks in supporting user queries. We envision the need for supporting multiple reasoning paradigms including relational query processing for SQL-type queries of the meta-data and meta-knowledge, reasoning about aggregation and generalization hierarchies in support of inheritance of attributes and methods associated with objects, reasoning using blackboard models of control in conjunction with multiple knowledge sources and an intelligent thesaurus for cooperative query formulation and optimization[5].

The ADKD should provide tools to analyze constraints and rules associated with an object-oriented schema. This is a relatively new area of research but it will prove very useful in handling complex object representations involving behavioral knowledge expressed as rules.

3.3.7 Knowledge Organization Services

Both knowledge and data objects must be indexed, catalogued, and cross-referenced for ease of storage and retrieval. For example in section 7 we present an object-oriented data model in which rules are defined as clauses in a first order logic. These rules are grouped into rule sets and are then associated with objects. Thus, there is a natural index between an object type and the rules associated with it. Conversely, given a rule one can easily determine to which object it refers. This object-oriented approach to indexing both objects and constraints has been proposed by Shepherd and Kerschberg [SK86b]. The ADKD must provide tools for the organization of both data, knowledge, meta-data and meta-knowledge.

---

3 In section 6 of this report we study schema evolution in object-oriented databases and propose a versioning approach that uses a truth maintenance system to assist the dictionary in support changes in schema definitions.

4 We have investigated and used two models that could be candidates for the ADKD object-oriented model. They are the Knowledge Data Model, and an object-oriented model that supports aggregation and generalization of object types, attributes and relationships among object types, and rules that can be associated to object types. These are presented in section 4 through 7 of this report.

5 Section 4 of this report studies the use of a blackboard model of control in conjunction with an intelligent thesaurus to coordinate query formulation and processing in a federated database architecture.

3.3.8 Translation and Mapping Services

The ADKD must support mappings from the internal knowledge/data representation to the structure, operations and constraints of the more established data models such as the relational, and network data models. The ADKD must also translate from one representation to another. This would allow the constituents of a federated database to communicate with one another to exchange data and possibly knowledge. In addition to supporting database data representations, the ADKD should support multiple viewpoints of data and knowledge.

3.3.9 Learning and Adaptation Services

In order for the ADKD to truly be active, it must provide services to users and applications that allow for the evolution of the objects stored in the Object Repository. In particular, data and knowledge schemata must be able to evolve as our real-world models and activities evolve. Therefore, the ADKD should provide tools for schema evolution, for knowledge refinement, and for "knowledge compilation" in the form of cases (see figure 3.3).

For example, the ADKD may assist in a problem-solving exercise involving multiple knowledge sources, and the results of the exercise will suggest a solution to the user[6]. The entire problem-solving exercise which includes intermediate results and the final solution can be added to a "case base" of such cases.

The AI discipline known as Machine Learning can play a major role in supporting evolution in databases. New tools and techniques are being developed to "discover" knowledge from data. One such activity at George Mason University is the INLEN project [KMK89] in which a relational database system and a rule-based system are connected through a loosely-coupled interface. Both systems have their full capabilities to support queries and reasoning, but in addition, a number of knowledge generation operators are available to learn new knowledge/data from the existing knowledge/data bases. This knowledge can be re-integrated into the knowledge/data bases and can also be used by decision-makers.

As an example, suppose INLEN were asked to discover knowledge about a database. It might consult the meta-data to verify the current data organization, and through one of its learning algorithms, propose a new functional dependency based on the actual data contained in the database. This new functional dependency is a constraint that may or may not hold over all time intervals because it depends on actual database instances rather than design knowledge. Therefore, the ADKD could be tasked to monitor the database regarding this functional dependency, and if it were violated during a database update, the Database Administrator could be notified.

The INLEN project is sponsored by DARPA. More research is needed to understand how discovered knowledge can be used effectively within the context of active data/knowledge dictionary systems.

---

6 Section 5 examines problem-solving using multiple knowledge sources: database meta-data, knowledge hierarchies, case and heuristics to solve complex network fault diagnosis problems. It also addresses knowledge compilation into cases.

3.3.10 ADKD Object Manager

All services and tools must communicate with the ADKD Object Manager to retrieve objects from the Object Repository. The Object Manager has meta-data and meta-knowledge of how the various objects and their representations are stored in the Object Repository. The Object Manager is much more than just an Object-Oriented Database Management System (OODBMS); it must provide for the management of heterogeneous and multiple types of objects produced by the various enterprise activities. A very important function of the Object Manager will be consistency checking and management of the diverse object types, their interrelationships and dependencies.

3.3.11 Object Repository

The Object Repository contains the ADKD knowledge and data regarding the enterprise's information architecture, including data definitions, knowledge representations, methodologies for data and knowledge base design, etc. We see the Object Repository as being implemented using an object-oriented database system. However, it is not clear if the entire ADKD should be implemented solely within the context of an OODBMS.

The ADKD most probably should be implemented as an expert database system architecture in which the knowledge-based components such as knowledge representations, blackboards, truth maintenance systems and case-based systems would be implemented as special tools that would access the Object Manager and Repository. Similarly, the special services such as Translation and Mapping, Knowledge Organization and Management, and Learning and Adaptation Services would be implemented as a suite of tools that could access the Object Repository for relevant information and could communicate through the Blackboard Model depicted in figure 3.3.

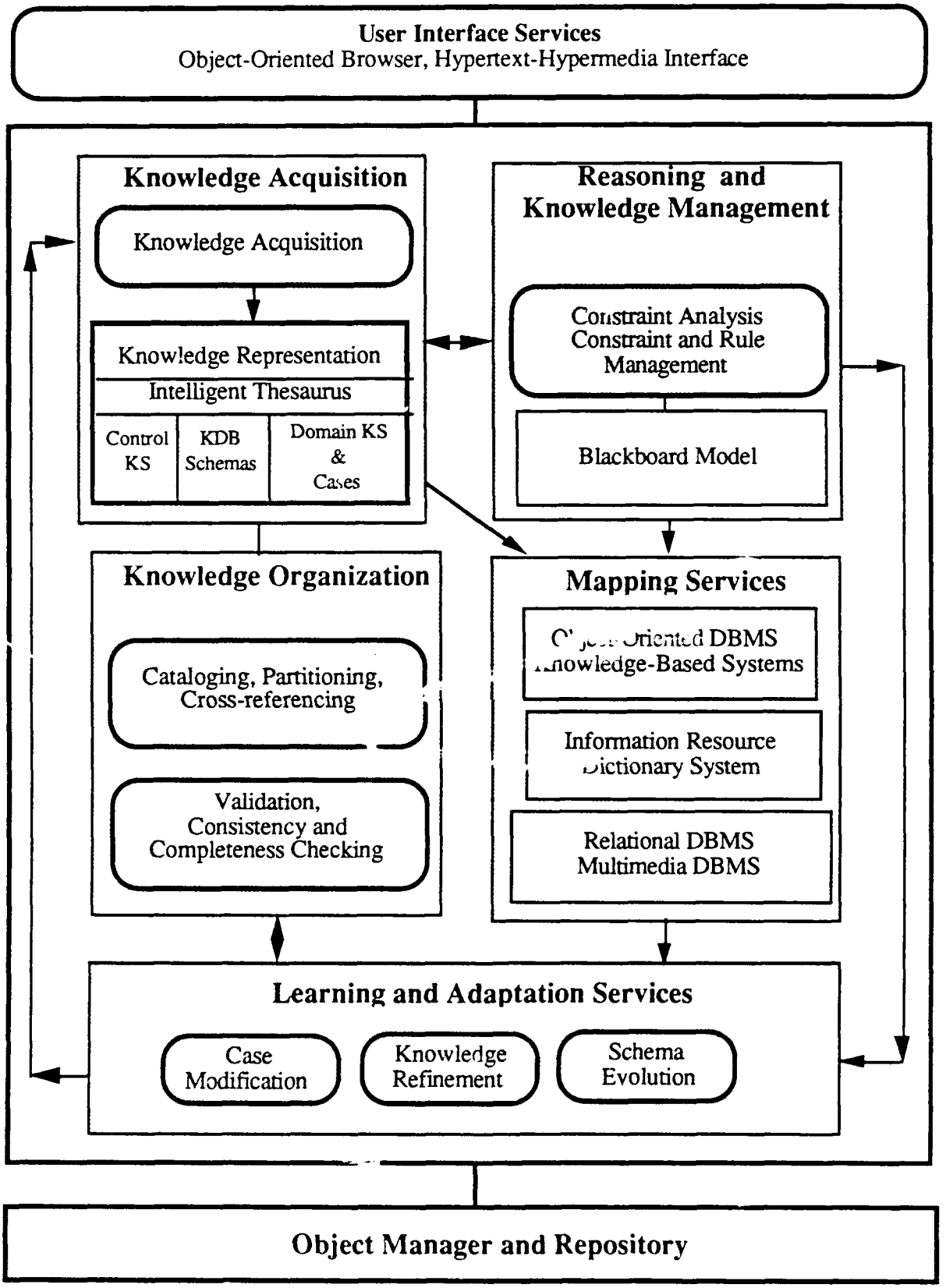


Figure 3.3 ADKD Tool Interrelationships

4 Active Query Processing in Heterogeneous Databases

Query processing comprises query decomposition, query optimization, and subquery processing. The general strategy for processing queries in a heterogeneous environment is for a global controller to decompose a global query (made by a local database) into appropriate local subqueries, to supervise the execution of the subqueries, and to integrate the subquery results into an answer for the requesting database. Optimization can occur anywhere in that process. An alternative strategy, appearing in the federated and interoperable approaches, is to have a single local query cause the generation of multiple queries to system databases without the aid of a global controller. These queries are based upon the local database's view of the external schema of each of the other databases. Results integration is left to the requesting database. Query optimization in both approaches is left up to the local user. In the interoperable approach, however, some optimization assistance is provided with its facility for allowing users to express "incomplete queries," i.e., queries without all equijoins specified [Lit87].

4.1 Query processing in heterogeneous databases — literature survey

There have been several approaches taken to solving the problem of accessing and interconnecting heterogeneous DBMSs [Car87, Dwy81, Fra87, G+84, HR87b, L+86, S+81, T+87a]. In general, these approaches have been to:

- Provide a standard view of single databases and supply access to one database at a time through a standard DML.
- Provide an integrated view of multiple heterogeneous databases and the capability to access and integrate data from several databases to answer a single query.
- Integrate data elements of interest from multiple databases into a single integrated database and access that database through a single DML.

The following paragraphs briefly describe some of the research efforts exemplifying these approaches. The efforts described are systems that are either under development, partially developed, or that are proposals and analyses for future systems.

4.1.1 MULTIBASE

Multibase is a system for integrating access to pre-existing, heterogeneous, distributed databases. Users view the database system through a single global schema (defined by the functional data model) and access data with a global query language called DAPLEX. The architecture of the Multibase system consists of two basic components: a schema design aid and a run-time query processing subsystem.

The schema design aid provides tools for the system database designer to design the global schema and to define a mapping from the local databases to the global schema and vice versa. The run-time query processing subsystem uses the mapping definition to translate global queries into local queries, ensuring that the local queries are executed correctly and efficiently by local DBMSs [S+81].

As shown in Figure 4.1, the architecture of the Multibase schema design aid has three levels of schemata: a global schema (GS) at the top level, an integration schema (IS) and one local schema

(LS) per local database at the middle level, and one local host schema (LHS) per local database at the bottom level. The local host schemata are the existing schemata defined in local data models and are used by the local DBMSs. Each of these LHSs is translated into an LS defined in the Functional Data Model. The IS describes a database that provides information used to resolve inconsistencies between the data of different local systems. The LS and IS are mapped into the GS. The GS allows users to pose queries against what appears to be a homogeneous and integrated database.

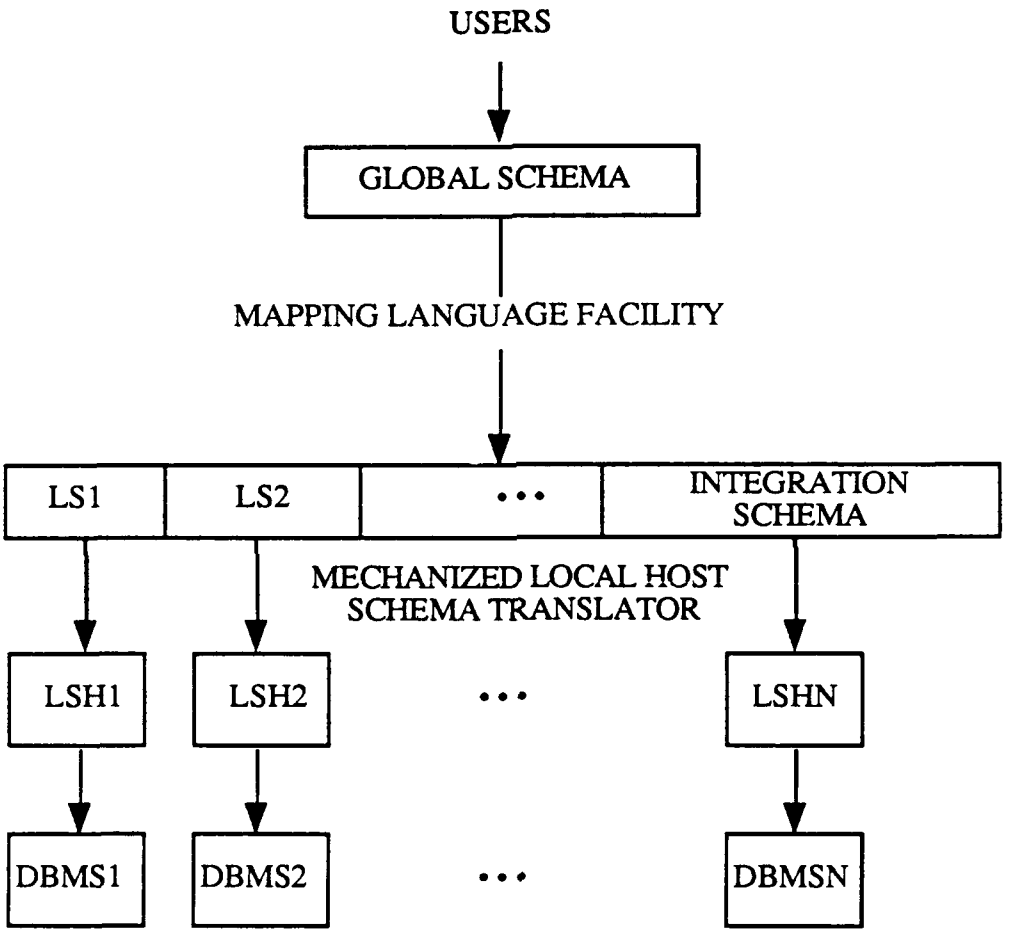


Figure 4.1 Multibase Schema Integration Architecture [S+81]

The architecture of the run-time query processing subsystem consists of a global database manager (which in turn consists of a query translator and a query processor), a local database interface (LDI) for each local DBMS, and the local DBMSs (see Figure 4.2). A user submits a query to the system (with DAPLEX) over the global schema. The query translator translates the global query over the global schema into a global query over the disjoint union of local schemata. The query processor decomposes the global query over the disjoint union of local schemata into individual local queries over local schemata. The query processor also does query optimization and coordinates the execution of local queries. The LDI translates local queries received from the query

processor into queries expressed in the local data query language and translates the results of the local queries into a format expected by the query processor, which integrates the results and provides an answer back to the user.

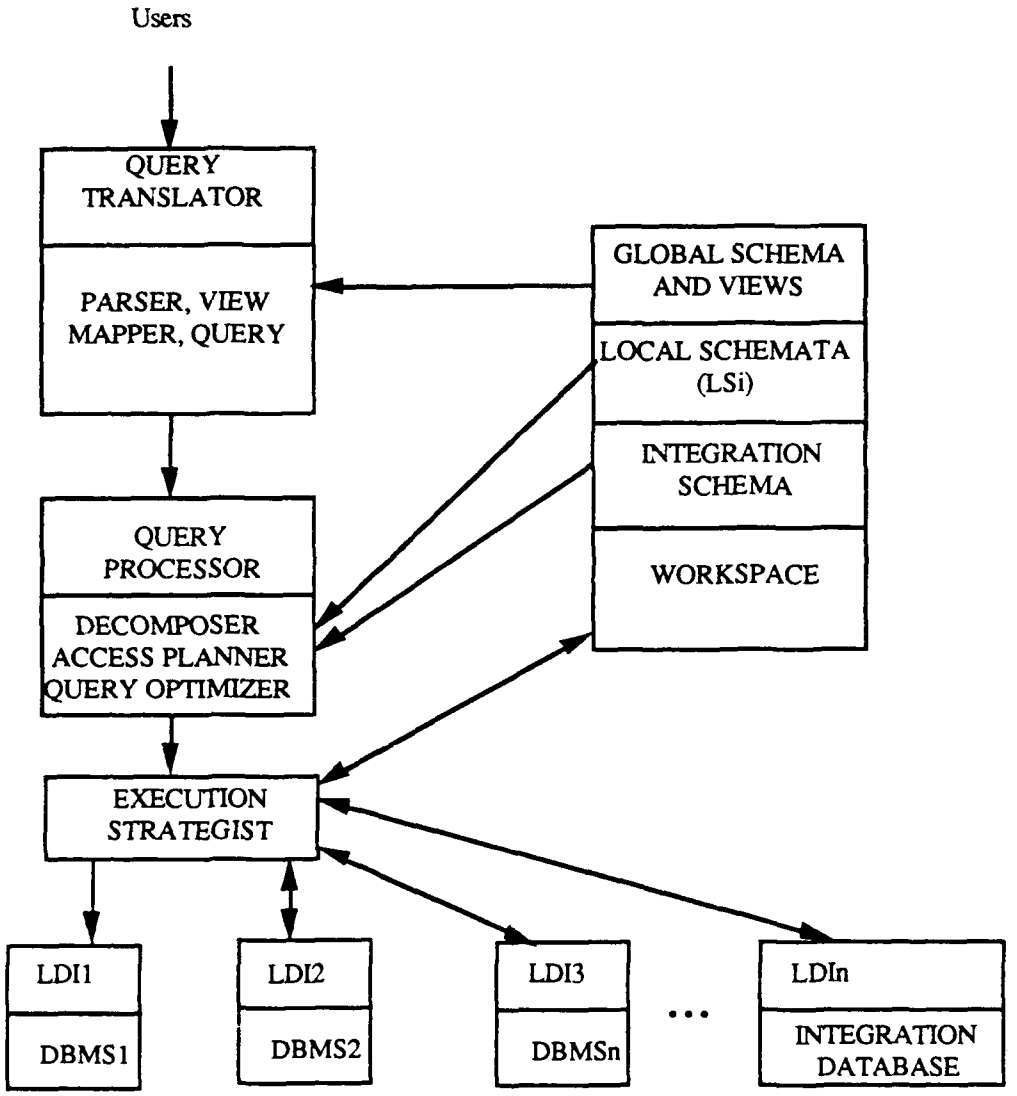


Figure 4.2 MULTIBASE Run Time Query Processing Subsystem [S+81]

4.1.2 MERMAID

Mermaid [T+87a], under development by Unisys Corporation, is a front-end to a network of multiple heterogeneous distributed databases (see Figure 4.3). The user interface provides a single integrated view of the underlying databases. This view is maintained in the global data dictionary which also stores the global schema.

Users access the HDBMSs using ANSI standard SQL or a common query language called ARIEL. The SQL or ARIEL query is translated into a global intermediate query language called DIL (Distributed Intermediate Language). If the DIL query can be answered at a single site, Mermaid

bypasses the distributed query processing and sends the query to the appropriate DBMS. Otherwise, the DIL query is decomposed into subqueries. The distributor, which contains the query optimizer and the system controller, develops a global subquery processing optimization scheme and routes the subqueries to the appropriate local DBMSs for data services. (Data services currently are limited to system-wide query and single database update.)

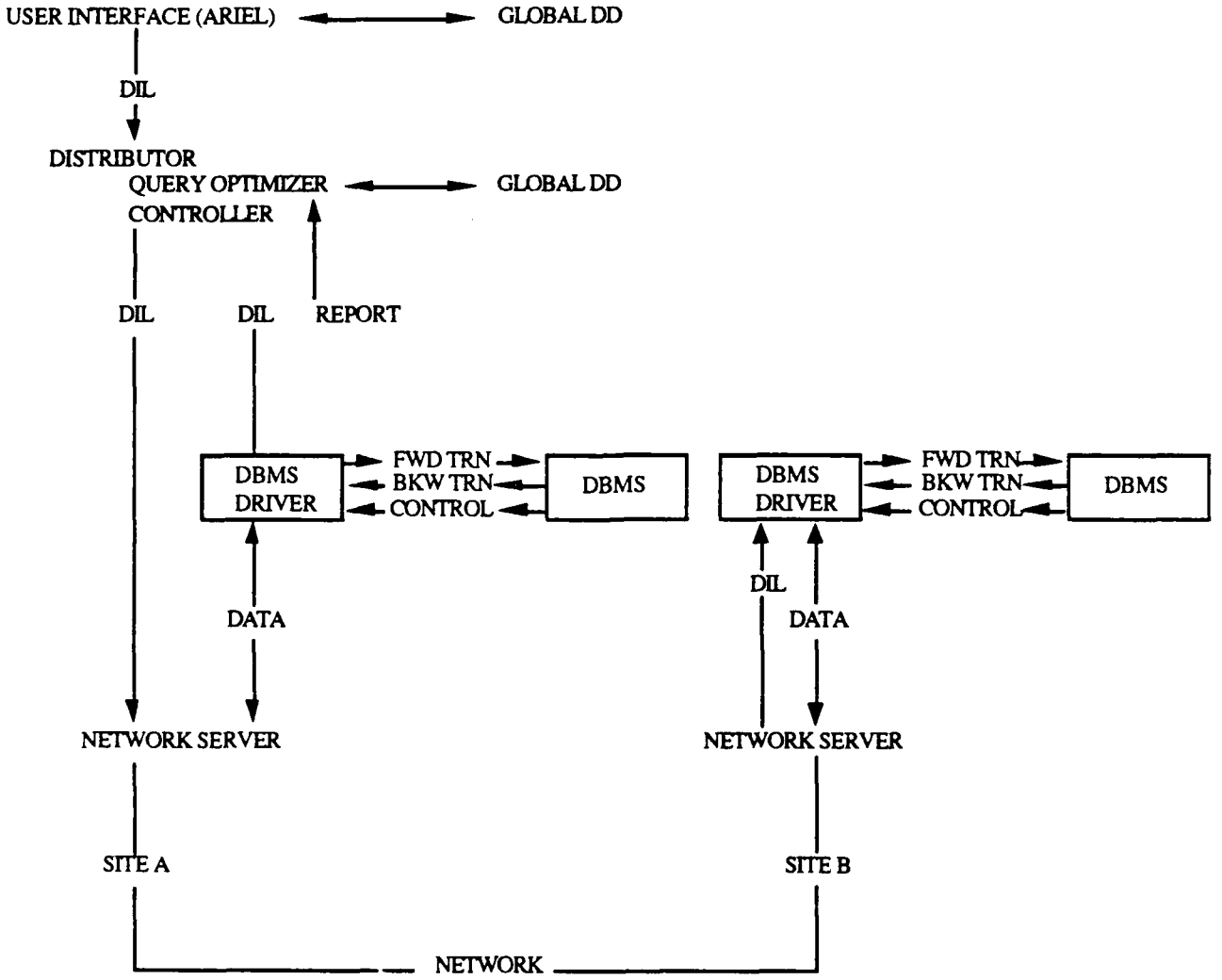


Figure 4.3 MERMAID Architecture [Y+85]

A DBMS driver is required at each site which contains a DBMS. The driver conducts the required query translation (from DIL to local DML and vice versa, denoted by "FWD TRN" and "BKW TRN" in Figure 4.3) and returns the response (if any) to the controller. The controller assembles the responses and directs the driver at the user's site to send the system response to the user interface (and thus the user).

A great deal of emphasis has been placed upon the development of algorithms for distributed query optimization [Y+85]. A semijoin algorithm tries to reduce relations as much as possible before sending them across the HDBMS network. A replicate algorithm tries to find an optimal set of sites at which the query can be executed in parallel. And a combined semijoin and replicate

algorithm seeks to find situations in which the semijoin and replicate algorithms can be exploited opportunistically.

Mermaid is a prototype that has been implemented on a VAX11/780 which is host to a back-end Britton-Lee IDM database machine, a Sun 120 with INGRES, a Sun 170 with INGRES, and a Sun 120 with Mistress [T+87a,T+87b,Tem88,Ull82].

4.1.3 MRDSM

Litwin and Abdelaziz have described the Multics Relational Data Store Multidatabase (MRDSM) [L+86] that features the notion of interoperability among HDBMSs. Unlike the architectures of Multibase and Mermaid, in which individual databases became components of an integrated system, the MRDSM interoperable architecture forces a degree of cooperation and partnership among databases. All participating DBMSs retain autonomy and control over their data. Participating system database administrators define interdatabase dependencies which in turn define interdatabase relationships with respect to the interdatabase integrity, privacy, and data meanings.

The general architecture of MRDSM is shown in Figure 4.4. MRDSM has no global schema. Databases become participants when their export schema is defined to MRDSM. An export schema may be a conceptual schema, a data model, or a database view schema. The export schema must conform to the relational model; the internal schema of the participant DBMS does not have that constraint, however.

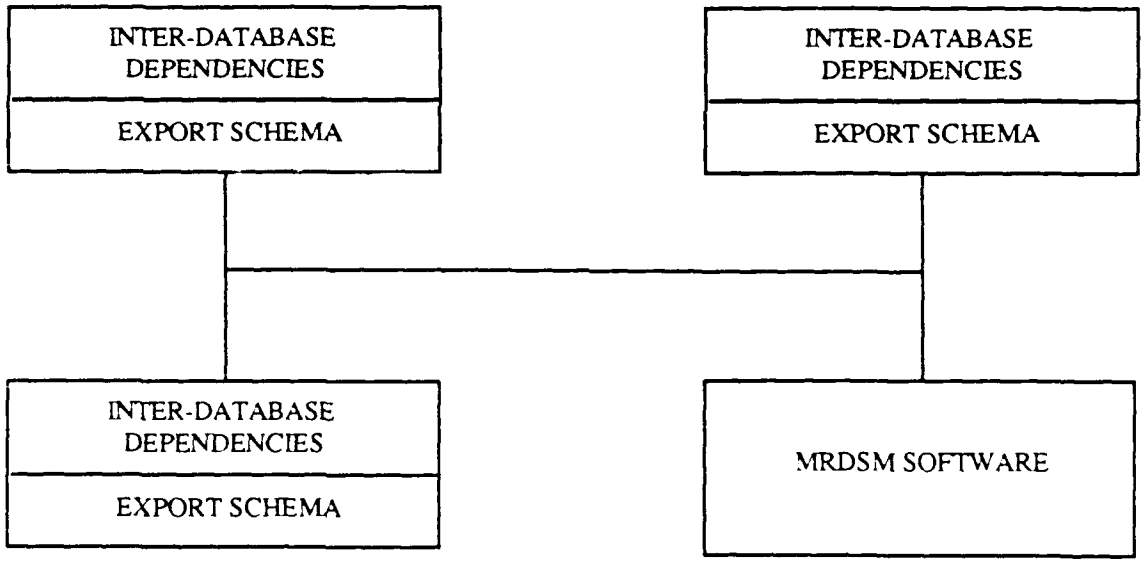


Figure 4.4 A Typical MRDSM Configuration

Users access system databases through the MRDSM data manipulation language, MDSL [Lit87]. Notably, both retrieval and update operations are allowed. However, users must know the contents of the participating databases to formulate MDSL queries. To assist users in query formulation, MRDSM provides commands to instantly display the export schemata of the relevant databases the users wish to access.

The goal of MRDSM is to allow users to formulate a system-wide query with a single statement. Multiple database queries are generated through manipulation dependencies. Manipulation

dependencies serve as an interdatabase message passing system. Manipulation dependencies route "triggered queries", which are queries (i.e., insertion, deletion, etc.) that precede or follow originating queries, throughout the system. Upon the occurrence of a query at a given database, manipulation dependencies trigger subsequent queries to predesignated system databases. These triggered queries may in turn become sources of other triggered queries.

MRDSM is being developed by INRIA (France). It is currently in the prototype stage and operates on a domain of multiple MRDS relational databases running on Honeywell systems.

4.1.4 KADBASE

KADBASE is a flexible, knowledge-based interface in which multiple expert systems and multiple databases can communicate as independent, self-descriptive components within an integrated, distributed engineering computing system [HR89] As illustrated in Figure 4.5, KADBASE is comprised of three basic components: a knowledge-based system interface (KBSI) for each knowledge-based system (KBS) in the overall system; a network data access manager (NDAM); and a knowledge-based database interface (KBDBI) for each database management system (DBMS) in the system. A KBSI possesses knowledge about the schema of the KBS context (data space) and uses that knowledge to perform semantic and syntactic translations for queries, updates and replies. A KBDBI acts as an intelligent front-end to a DBMS. It possesses knowledge about a specific local database schema and DML. The KBDBI uses that knowledge to perform semantic and syntactic translations for queries, updates and replies. The NDAM receives global requests for data services from KBSs, decomposes the requests into subrequests, and sends the subrequests to the appropriate KBDBIs. It receives local replies from KBDBIs, combines these local replies into a global reply, and forwards the global reply to the requesting KBS via the KBS's KBDI.

For example, if a request for data is issued by a KBS, the KBS's KBSI translates the request from the data manipulation language (DML) embedded within the KBS to a global DML. The KBSI forwards the query to the NDAM. The NDAM locates sources for the data referenced in the request and decomposes the global query into a set of subqueries or updates to individual target databases. The subqueries are then sent to the appropriate KBDBIs for processing. Upon receiving the data from their supported DBMS, each KBDBI returns the data to the NDAM. The NDAM aggregates the individual replies and forwards the global reply to the requesting KBS via the KBS's KBSI.

The frame data structure is used to represent the KADBASE global schema and local schemata. Mapping is done from the local schema to a local frame-based schema, and from the local frame-based schema to a frame-based global schema by local and global KSs respectively.

The global data source mapping KS essentially plays the part of the data dictionary/directory -- a table lookup exercise. Local mapping is more sophisticated, in that the LFBS may contain information about semantic relationships between entities not found in the underlying data models.

One of the features of KADBASE is that it allows updates. Although not specified in the literature, it appears that concurrency problems are avoided by insisting that all updates are conducted/coordinated by the NDAM. While this approach does preclude the need to address concurrency issues, it is not a particularly flexible approach in that it destroys all autonomy for system DBMSs.

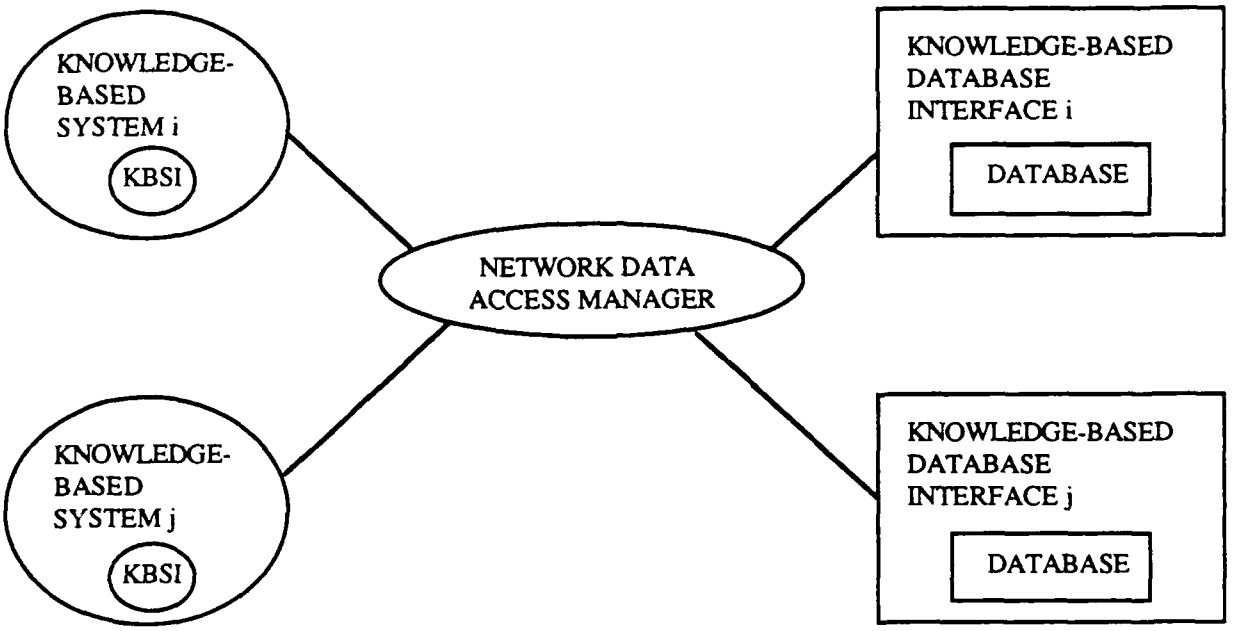


Figure 4.5 The KADBASE Architecture [HR89]

Although data service requests in KADBASE are made by KBSs rather than users, one can think of these KBSs as agents for users. However, the queries/updates that the KBSs generate are essentially pre-defined in that they are limited by the scope of the application. Therefore, there is no need to offer assistance in query formulation.

The KADBASE prototype and its demonstration applications (SPEX, a structural component design expert system, and HICOST, an expert cost estimator for detailed building designs) have been implemented on a VAX 11/750 and several MicroVAXs. The KADBASE sample databases are under INGRES [HR87b,HR89].

4.1.5 ANSWER

Honeywell's ANSWER (Army's Nonprogrammer System for Working Encyclopedia Requests) prototype assists untrained users in learning about the contents of an organization's databases (assumed to be distributed and heterogeneous). Further, it helps them formulate queries of those databases [D+88]. The ANSWER architecture is shown in Figure 4.6.

The Encyclopedia module contains high-level information representing an organization's activities ("enterprise objects"), mid-level information representing a global database schemata, low-level information representing "local" database schemata, and information mapping local objects to global objects to enterprise objects. The mapping information is the by-product of schema integration occurring during Database Registration.

Database Registration allows the Encyclopedia Administrator to establish the semantic and syntactic mappings required to make a new database known to the Encyclopedia. (Note this is performed manually.) These mappings are the result of a four-phased schema integration process in which: (1) component schemata are specified using a common data model; (2) schemata incompatibilities are resolved; (3) object classes and relationship sets (i.e., object classes whose domains are equal, contained, disjoint but integrable, etc.) are established; and (4) object classes and relationship sets

are integrated. A novel feature incorporated into the Database Registration module is its "knowledge extraction facility." The knowledge extraction facility is an expert system shell (i.e., a user interface and inference mechanism) that aids in schemata integration. It automatically extracts local schemata data element relationships from the Encyclopedia.

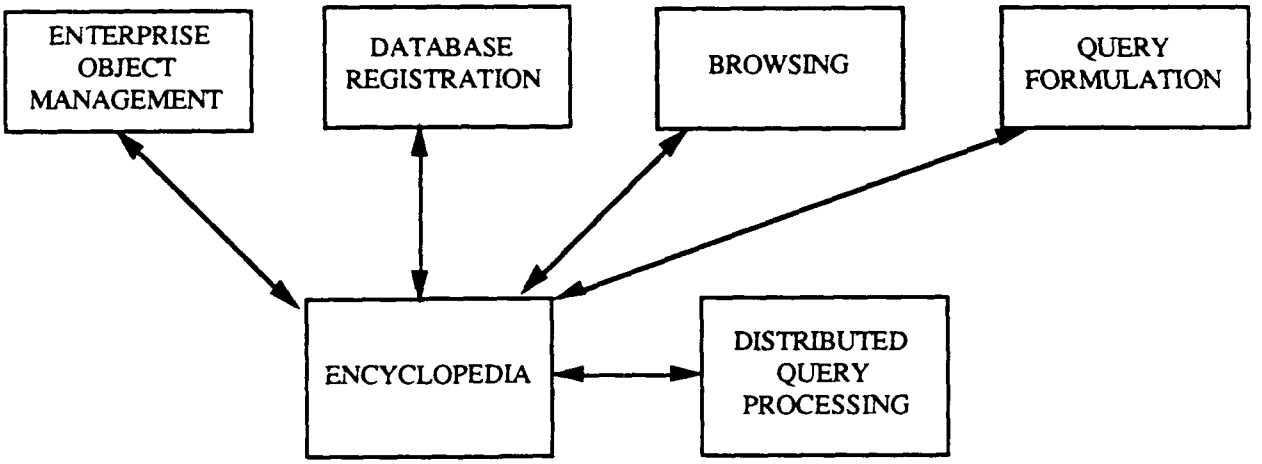


Figure 4.6 The ANSWER Architecture [D+88]

The Browsing module presents users with graphical representations of Encyclopedia object names and relationships. It presents menus to the user that allow the user to select object types for access, to scroll or page through object types or instances of the object types, and to get more detail about specific object types or instances (zooming).

The Query Formulation module is somewhat of a misnomer. As described in [D+88], assistance in query formulation is offered in two ways. Firstly by enabling the user to browse the Encyclopedia (through the Browse module). And secondly, assistance is rendered by offering users either a syntax diagram [L+84] or an interactive SQL syntax error detector, or both. This second feature of the Query Formulation module does not appear to have been implemented as of this writing. Another unimplemented module is the Distributed Query Processing module. The developers plan on using an existing distributed DBMS, such as Honeywell's Distributed Database Testbed System, to implement distributed query processing. The query processing goal of ANSWER is to be able to automatically formulate and optimize queries that can be executed on distributed HDBMSs. It should be noted, however, that the authors' definition of heterogeneous only extends to HDBMSs supporting SQL.

The initial prototype of ANSWER is currently being implemented. The goals for the initial prototype are to allow users to maintain objects corresponding to high level concepts in the enterprise model, register new databases with the Encyclopedia, browse through the enterprise, derived and schema objects, and formulate SQL queries against schema objects.

4.1.6 ACDB INTERFACE

Semmel et al., [S+87] have proposed an architecture for an intelligent interface to the U.S. Army's corporate database (ACDB). The objective of this research is to construct an interface that is able to help casual users query the Army corporate database and assist those users in the analysis of query results. The interface has been designed to support a consolidated SQL database that has been composed from data elements of various production databases. As one can discern from

Figure 4.7, it is being designed to incorporate practically every feature imaginable (the "Swiss Army knife" of intelligent DBMS interfaces).

The function of the Intelligent Navigational Assistant (INA) is to assist users in constructing queries and analyzing results after query execution. It is responsible for constructing SQL queries (to include identifying syntactic errors and semantic inconsistencies) and analyzing query results (to include security analysis i.e., the "Ancillary Functions" of Figure 4.7).

The Knowledge Base is the primary knowledge support structure. It stores user vocabulary knowledge, query formulation knowledge, data model knowledge, update and validation criteria knowledge, and ancillary knowledge (such as security knowledge).

The Update Manager is designed to monitor the transfer of data from the Army's production databases to the corporate database. Temporal information is recorded, which is used by the INA to determine potential temporal inconsistencies. The Update Manager also insures that updates to the corporate database are not inconsistent.

The System Manager is designed to manage the total AI interface environment. Its task is to provide knowledge structure maintenance capabilities so that the knowledge base and augmentation files may be reviewed or modified by the AI interface administrators. It also incorporates tools that enable it to monitor and accrue statistics concerning interface performance and user utilization.

The Augmentation Files contain knowledge of user profiles, temporal and inconsistent data files, a user query history file, and security profiles.

A preliminary prototype was built at Johns Hopkins Applied Physics Lab. The prototype is PC-based and allows users to formulate SQL queries. The prototype will guarantee syntactic correctness but not semantic correctness. The project is currently inactive; the Army withdrew its funding because of fiscal constraints.

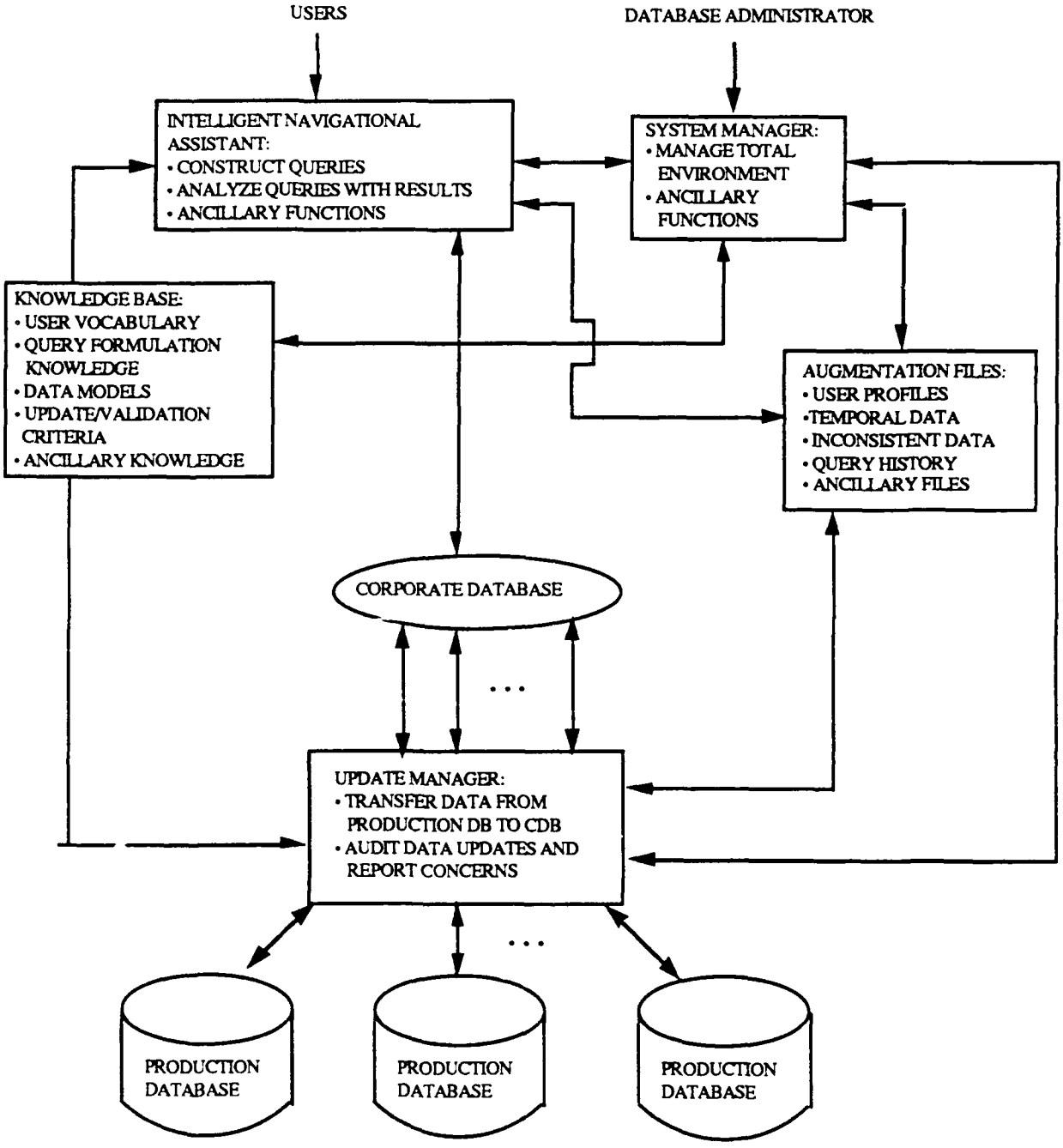


Figure 4.7 Functional Architecture of the ACDB AI Interface

4.1.7 SUMMARY

The systems reviewed in the preceding sections are all designed to address the problem of integrating existing databases. Table 4.1 juxtaposes these systems against the features that are of particular interest to this research. While much of this information has been presented earlier, certain aspects warrant further discussion.

The majority of the systems incorporate the relational model as their global data model. The stated rationale behind this choice is the relational model's ubiquity. While this property may reduce the time and complexity associated with implementing a heterogeneous system (i.e., there will be less translation involved since most DBMSs are relational), it does not reduce the fact that the relational model is very restrictive in its ability to accurately represent an enterprise. Semantic data models, such as the functional data model (used in Multibase) and the entity-relationship model (used in ANSWER), allow the enterprise to be represented in a more flexible and powerful way than through traditional models. However, these models do not afford a method to incorporate knowledge and data in a unified manner. This is a disadvantage when attempting to implement an intelligent interface.

The KADBASE frame data model (which is also semantic data model based) is analogous to the Knowledge/Data Model in that each frame-based schema incorporates knowledge about the local schema it describes. However, it falls short of incorporating all of the important data modeling primitives of the KDM. Further, it shares the property of semantic data models in which objects are essentially static, rather than being viewed to have intrinsic behavior (such as having the ability to send and receive messages) as in the AI world.

While the majority of systems reviewed make use of data dictionaries, there has been little effort to develop and incorporate an integrated and active global thesaurus. The ANSWER system thesaurus can be regarded as a thesaurus only in a very weak sense. That is, users can browse the thesaurus to determine the broader and narrower terms. However, the thesaurus fails to provide the necessary functionality for an active thesaurus as defined by McCarthy [McC88].. Further, it is passive in nature in that users must access it, rather than it playing a proactive role with the user.

The proactive role of the thesaurus centers on providing users with query formulation assistance. These systems are weak in this regard. Most have no help with query formulation. The Mermaid user interface provides standard DBMS interface support but offers assistance only in editing queries. As mentioned in the review of ANSWER, the notion of query formulation help is deceiving in that users must guide themselves through the database content familiarization process. The ACDB system is the only system in which active assistance with query formulation is proposed.

A final point on Table 4.1 is that the notion of an intelligent interface appears in none of the prototype systems. It has been proposed for the ACDB system, but has not been implemented in the preliminary prototype.

Table 4.1 Summary of Pertinent System Features

Feature	System MULTIBASE	MERMAID	MRDSM	KADBASE	ANSWER	ACDB
Global Data Model	Functional	Relational	Relational	Frame	Entity/ Relationship	Relational
Global DML	DAPLEX	SQL or ARIEL	MDSL	Frame	SQL	SQL
Global Thesaurus or Data Dictionary	No (Schema Integration)	Data Dictionary	No	Data Dictionary	Thesaurus	Data Dictionary
Global Schema Construction	Manual	Manual	N/A	Manual	Knowledge Extraction Facility	Manual
Global Updates	No	SQL Database	Yes	Yes	No	No
Query Formulation Help	No	Syntax Only	No	No	Yes	Yes
Query Optimization Facility	Limited	Yes	No	No	Planned	No
Intelligent Interface	No	No	No	No	No	Yes

4.2 The knowledge/data model

The Knowledge/Data Model is an extension of the semantic data model and draws heavily upon the features of the functional data model [SK77, KP76] and the object-oriented paradigm [Z+86]. Developed by Kerschberg and Potter, the KDM belongs to a class of hyper-semantic data models, which facilitate the incorporation of knowledge in the form of heuristics, uncertainty, constraints and other AI concepts, together with object-oriented concepts found in semantic data models [PK89]. It contains modeling features that allow the semantics of an enterprise to be captured. These semantics include data semantics, as captured by semantic data models and knowledge semantics, as captured in knowledge-based systems.

The KDM specification language is known as the Knowledge Data Language (KDL). The KDL maintains a knowledge base that contains the system's domain specification knowledge, the KDM specification knowledge, the KDM meta-knowledge, and the domain KDM schema. The domain specification knowledge, the KDM specification knowledge and the KDM meta-knowledge are

referred to as the control knowledge. The control knowledge is used for manipulating and handling the KDM primitives (described later), domain meta-data, and knowledge. The domain KDM scheme is used to define and maintain the domain database. Knowledge, rules,and data are represented in the KDM as <attribute, object, value> (AOV) triples. Objects are related to one another in a semantic net through six KDM relationship primitives. These are generalization, classification, aggregation, membership, constraint/heuristic, and temporal. They are described briefly below:

Generalization: Generalization provides the facility in the KDM to group similar objects into a more general or higher-level object. This is done by means of the "is-a" relationship. This generalization hierarchy establishes the inheritance mechanism.

Classification: Classification provides a means whereby specific object instances can be considered to a higher-level object-type (an object-type is a collection of similar objects). This is done through the use of the "is-instance-of" relationship.

Aggregation: Aggregation is an abstraction mechanism where an object is related to the components that make it up via the "is-part-of" relationship.

Membership: Membership is an abstraction mechanism that specifically supports the "is-a-member-of" relationship. The underlying notion of the membership relationship is the collection of objects into a set-type object.

Temporal: The temporal relationship primitive relates object-types by means of synchronous and asynchronous object linkages.

Constraint/Heuristic: This primitive is used to place a constraint on some aspect of an object, operation, or relationship via the "is-constraint-on" relationship, or to attach an heuristic via the "is-heuristic-on" relationship.

These primitives give a systems designer the abstraction mechanisms requisite for flexible knowledge and data modeling. The generalization primitive allows classes to be organized into hierarchies of superclasses and subclasses. The classification primitive allows individual objects to be classified as instances of a class. The aggregation primitive allows the formation of complex object-types from a number of simpler object-types. The membership primitive allows objects or classes to be grouped as members of some higher level set of objects. Constraints express the semantic relationships that exist between the extensional data of the KDM schema, and heuristics can be used to derive intensional data. Lastly, temporal primitives express the *temporal* relationships that exist between object-types representing tasks or events.

4.3 The knowledge/data language

Associated with the KDM is a schema specification language called the Knowledge/Data Language (KDL). Figure 4.8 shows a general template for an object-type (class) specification employing the KDL. KDL reserved words are shown in uppercase letters. Identifiers shown in lowercase letters are place holders for user input. Optional items in the template are enclosed in square brackets, and at least one of each of the items contained in curly brackets must be part of the specification.

```
OBJECT-TYPE: object-type-name HAS
[ATTRIBUTES:
    {attribute-name:
      [SET OF | LIST OF] value-type
      [COMPOSED OF {attribute-name,}]
    [WITH CONSTRAINTS {predicate,}]
    [WITH HEURISTICS {rule,}];}]
[SUBTYPES:
    {object-type-name,}]
[SUPERTYPES:
    {object-type-name,}]
[CONSTRAINTS:
    {predicate,}]
[HEURISTICS:
    {rule,}]
[SUCCESSORS:
    {object-type-name,}]
[PREDECESSORS:
    {object-type-name,}]
[CONCURRENTS:
    {object-type-name,}]
[MEMBERS:
    {member-name: member-type,}]
[INSTANCES:
    {instance,}]
END-OBJECT-TYPE
```

Figure 4.8 KDL object-type specification template

Example: Consider a hypothetical Air Force C^2 example in which one models the concepts and relationships associated with an aircraft squadron. The object types are Squadrons, Aircraft, Pilots, Missions, and Sorties. A squadron has pilots, schedules aircraft, and conducts missions consisting of several sorties. These are depicted in Figure 4.9 using the KDM and its graph representation formalism which is based on the Functional Data Model. Single-, double-, and triple-headed arrows denote single-valued, set-valued and list-valued functions, respectively. The rectangles denote object-types. Here we note that some objects are standard while others are inferred (virtual) (e.g., Aces and the Strength of a Squadron). In this particular example the concept of a squadron's strength, a function, is based on the number of missions flown, aircraft scheduled, the number of pilots, and the ratio of aces to regular pilots. The English statement for the squadron's strength is:

> A squadron's strength is "Strong" if the squadron conducts at least 50 missions per month, schedules more than 20 aircraft, has at least 10 pilots, and the ratio of Aces to pilots is at least 0.25.

The associated heuristic, SH, for the strength attribute is:

```
FOR EACH s in Squadron
SH-C1: IF COUNT(conducts (s)) ≥ 50 AND
SH-C2: IF COUNT(has-pilots(s)) ≥ 10 AND
SH-C3: IF COUNT(schedules(s)) > 20 AND
SH-C4: IF RATIO(COUNT(aces(s) TO COUNT(pilots (s))) ≥ 0.25
       THEN
R1:    strength(s) = "strong"
```

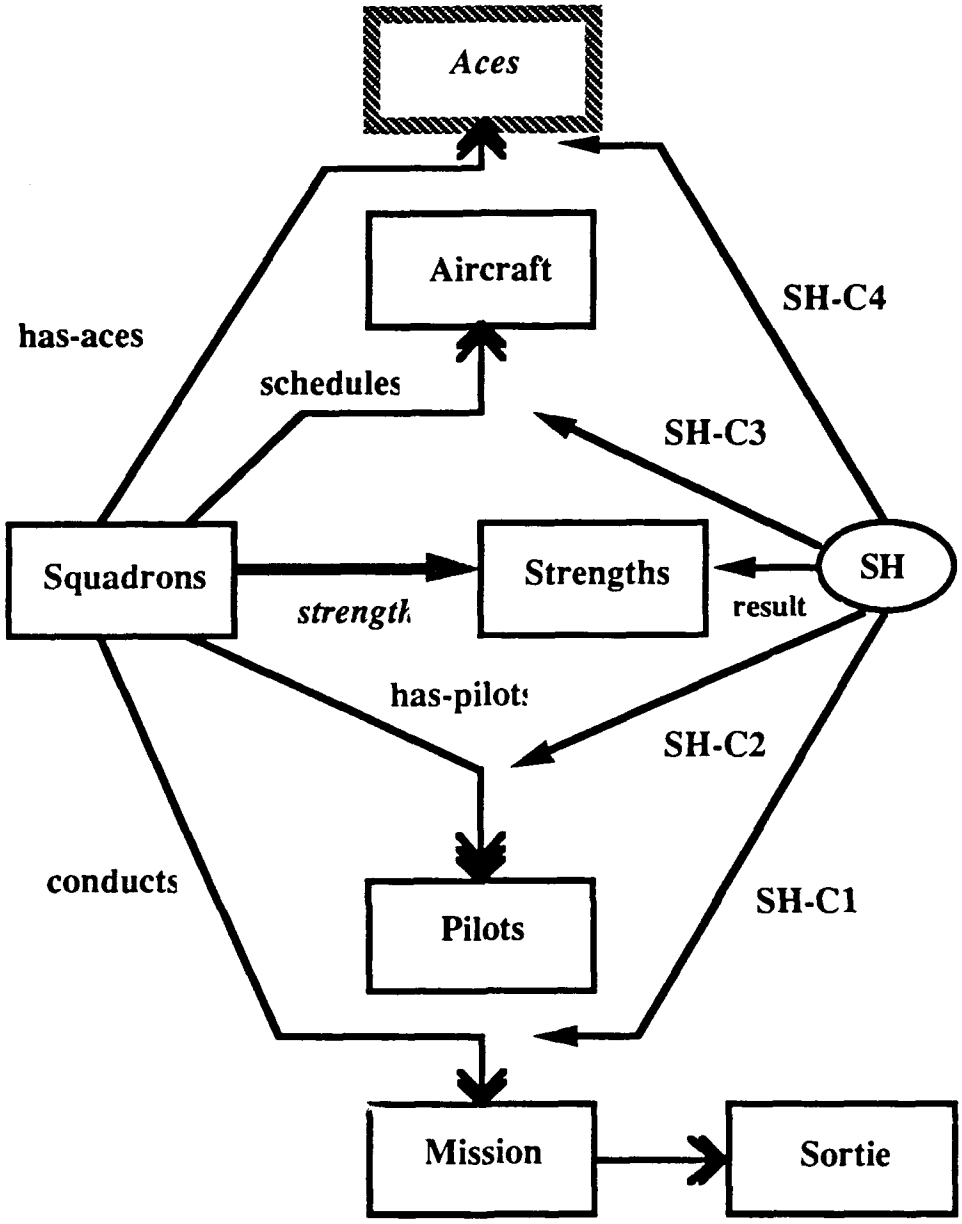


Figure 4.9 An Air Force C^2 Example

Note that the virtual type *Aces* can also be defined in terms of an heuristic that defines those pilots qualifying to be considered Aces. In processing the strength heuristic, the KDM processor would first evaluate the Aces (possibly) through inference and then evaluate the strength heuristic.

An important issue is that the KDM's constraint and heuristic primitives allow knowledge-based concepts to be specified as easily as database structures. This capability is similar to database views, but the view is very tightly associated with the schema. In addition, views involving views are quite natural, e.g., the Aces virtual type used in the strength heuristic.

In working with the KDL, we have recognized its weakness in modeling the behavior aspects of a domain, because it does not associate operations with objects. Potter's continued, independent work with the KDM describes a variant of the KDL called the Active KDL which does associate

operations with objects (Potter89). This the KDM and the Active KDL are being implemented by Potter's group at the University of Georgia.

The advantage of using the KDM

The advantage of using a hyper-semantic data model, such as the KDM, is that it allows a system designer to represent both knowledge and data in a unified, object-oriented structure. By associating domain knowledge and heuristics with objects, one can model the behavior of objects as well as the dynamic relationships with other objects. The significant features of a hyper-semantic data model are:

- The incorporation of heuristics to model inferential relationships.
- The capability to organize these heuristics and to associate them with specific items involved in the inferential relationships.
- The capability to incorporate heuristics and constraints in a tightly coupled (unified) manner.
- The ability to define inferred (virtual) objects.
- A unified representational formalism for knowledge and data.
- A mechanism that allows for abstract knowledge typing (handling rules/constraints as objects).

Heuristic relationships among database objects and object-types can be coded which give the knowledge/data administrator a facility for defining object restrictions, specifications and inferred information regarding the nature of an object. This allows one to compare the meaning of system objects to user query objects in order to determine the user's meaning of those objects (a very useful capability for improving query specifications). Users can be allowed to express queries in a more general manner because the KDM formalism allows the semantics of database terms to be captured.

The majority of the systems surveyed incorporate the relational model as their global data model. The stated rationale behind this choice is the relational model's ubiquity. While this property may reduce the time and complexity associated with implementing a heterogeneous system (i.e., there will be less translation involved since most DBMSs are relational), it does not reduce the fact that the relational model is very restrictive in its ability to accurately represent an enterprise. Semantic data models, such as the functional data model (used in Multibase) and the entity-relationship model (used in ANSWER), allow the enterprise to be represented in a more flexible and powerful way than through traditional models. However, these models do not afford a method to incorporate knowledge and data in a unified manner. This is a disadvantage when attempting to implement an intelligent interface.

The KADBASE frame data model (which is also semantic data model based) is analogous to the Knowledge/Data Model in that each frame-based schema incorporates knowledge about the local schema it describes. However, it falls short of incorporating all of the important data modeling primitives of the KDM. Further, it shares the property of semantic data models in which objects are essentially static, rather than being viewed to have intrinsic behavior (such as having the ability to send and receive messages) as in the AI world.

4.4 Intelligent query processing using the Intelligent Heterogeneous Autonomous Database architecture (InHead)

The InHead approach is to extend the state of the art in heterogeneous DBMS interface technology by integrating Artificial Intelligence (AI) problem-solving techniques with advanced semantic data modeling techniques. The approach draws upon the flexible and opportunistic problem-solving capability of blackboard architectures, and the expressive power of the Knowledge/Data Model (KDM) which allows both knowledge and data to be represented in a unified data knowledge representation.

In contrast to the sequential and hierarchical nature of standard interfaces to heterogeneous databases, e.g., Mermaid, Multibase and MRDSM, the InHead system incorporates an object-oriented Knowledge/Data Model, the KDM, and knowledge sources (KSs) possessing global and local domain expertise.

These KSs work together to provide users with *simultaneous, multiple* viewpoints of the system at varying levels of abstraction. One KS can be looking at a user's problem from a global perspective, while another can be viewing it in terms of a local database. In this way users can more fully determine system-wide data relationships.

Furthermore, in decomposing user queries, the opportunistic nature of the blackboard provides users with responses that are not only more complete (by KSs being activated on partial and incremental information), but reflect a deeper understanding of the user's desires. Also, it is envisioned that one KS will be a "User Model" and agent that contains a characterization of the user's intentions or goals during the query sessions.

A novel feature of InHead is a *global thesaurus* KS. The thesaurus addresses semantic heterogeneity issues in which data items may be similarly named, are related, are a subclass of, and/or are a superclass of data items located elsewhere within the system's databases. In this regard, the thesaurus plays the traditional role of data dictionary. What separates this approach from others is that the thesaurus takes on a new role: it ***actively*** works with users to reformulate queries based on knowledge associated with terms and their usage in the local databases. Figure 4.10 illustrates the architecture.

Conceptually, the system works as follows. Upon accepting a user query, the system controller consults the global thesaurus. If the solution to the query is obvious, the controller acts upon the query by sending it to the appropriate database(s). If not, the controller posts the query on the blackboard. That enlists the aid of the KSs, which are domain experts over their respective DBMSs. The KSs cooperatively try to find a solution to the query. If no solution can be found, a request to the user for clarification or further information is generated.

Thus the InHead thesaurus is used to perform semantic query processing of the user's original request before submitting the reformulated query to the local databases for processing. The controller then conducts the necessary query translation and optimization, sends the query (or subqueries) to the appropriate database(s), integrates the query results (if required), and provides the answer to the user.

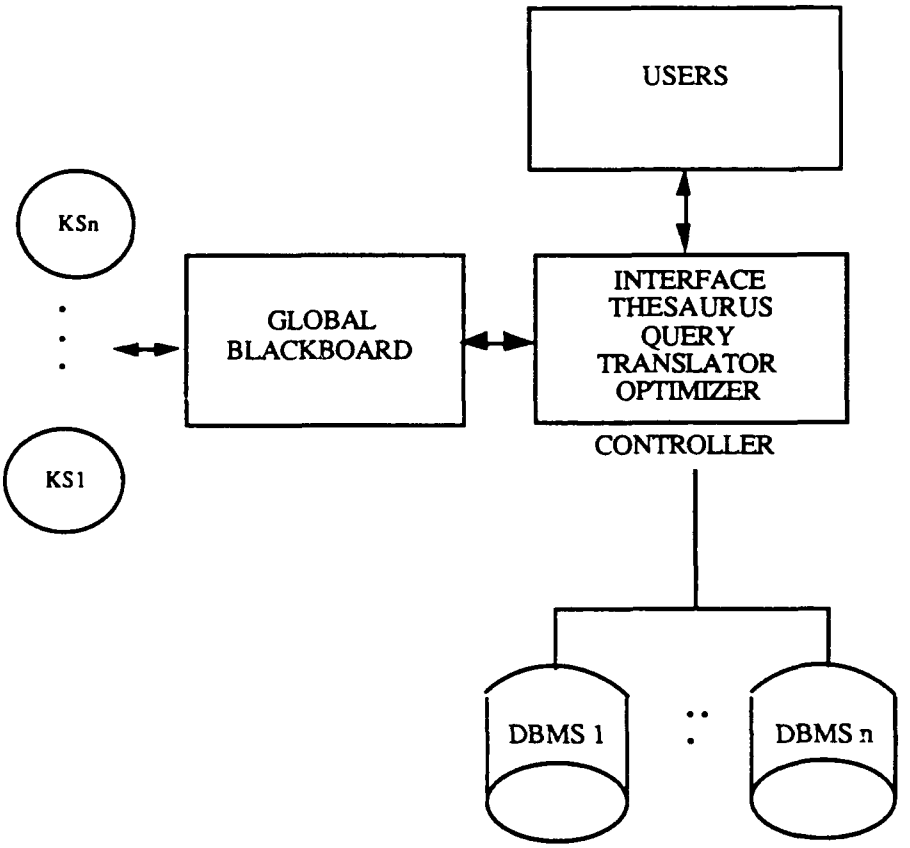


Figure 4.10 The Intelligent Heterogeneous Database (InHead) Architecture

4.4.1 The active and intelligent global thesaurus

Conventional thesaurus functions include meta-data management, data descriptions, descriptions of the relationships between the terms used to describe meta-data, as well as term definitions and descriptions. Thesauri become *active* when they are extended to provide functions such as: (1) validating and performing consistency checks on input to the thesaurus itself; (2) indexing and converting data values; (3) automatically translating queries using different variants of names; and (4) actively participating in on-line help (i.e., offer suggestions). By incorporating knowledge (in the form of heuristics and constraints attached to objects) into the thesaurus, we have not only made the thesaurus more active, but intelligent! Our thesaurus can act both as a repository of knowledge of data-item terms and of their usage, and as an active participant in formulating improved query specifications (i.e., by providing global data-item definitions and locations).

In essence, our active and intelligent global thesaurus provides the strategic problem-solving knowledge required to control semantic heterogeneity. The thesaurus is the ideal KS to resolve issues of semantic heterogeneity. Local database terms, concepts, relationships, constraints and operations that have semantic variants can be "encapsulated" into an abstract object type with sufficient domain knowledge to be able to translate and interpret the appropriate meaning for an object. The active thesaurus objects can invoke appropriate methods to present the proper views to the various local databases and to translate the global concept to the corresponding terms for the local KS.

The thesaurus could be used to incorporate newly discovered knowledge that might result from an examination and integration of KSs associated with each local site and with the overall problem-solving interactions among the sites in responding to semantically ambiguous queries. This knowledge could then be used as strategic case-based knowledge in future problem-solving exercises. The configuration management issues associated with knowledge evolution are topics of current research in InHead.

4.4.2 A troop capacity example query

To more fully illustrate the research approach, the following example is presented. It shows how the InHead KSs interact on a blackboard to answer a user query. The example also serves to highlight the aforementioned research issues. The existence of the intelligent interface and of the databases it supports is assumed. The schemata for these databases appear in Figure 4.11.

ARMY AIRCRAFT DATABASE:
FXD_WING(SER#,MOD#,CAPACITY)
ROTARY(SER#,MOD#,LIFT)

NAVY AIRCRAFT DATABASE:
BOMBERS(SER#,MOD#,CAPACITY,RANGE,CREW_SZ)
FIGHTERS(SER#,MOD#,SPEED,CREW_SZ)
UTILITY(SER#,MOD#,CONFIG)

AIR FORCE AIRCRAFT DATABASE:
TRANSPORT(SER#,MOD#,CAPACITY,SPEED,RANGE)
BOMBERS(SER#,MOD#,TONNAGE,SPEED,RANGE)
FIGHTERS(SER#,MOD#,SPEED,RANGE)

Figure 4.11 System Database Schemata

Figure 4.12 shows the translation of the Figure 4.11 schemata into corresponding KDM schemata. For brevity, Figure 4.12 shows only one of the object types (which corresponds to the above relations) for each of the KSs. Note the natural way in which the KDM paradigm depicts the objects, operations, and relationships being represented in the underlying system databases.

Suppose that a Pentagon official wished to determine the troop carrying capacity of all military aircraft. Using the InHead system the Pentagon action officer would pose the following simple SQL query to the interface:

SELECT capacity FROM aircraft WHERE (capacity = troop)

There is a great deal of ambiguity in this query. Firstly, the meaning of capacity. Should cargo capacity that can be converted to troop capacity be included? Secondly, one needs to know what is meant by aircraft. Are all aircraft included as the queries implies? Did the user really mean to include dirigibles, trainers, and spy planes?

The InHead interface begins the reformulation process by placing the query on the blackboard. The global thesaurus knowledge source (KS) recognizes the object type aircraft and places the aircraft object subclasses that it knows about (i.e., bombers, fighters, transports, utility, fxd_wing, and rotary) on the blackboard. Those subclasses are then returned to the user for more specificity (including prompts for universal and existential quantification).

GLOBAL THESAURUS KS:

```
OBJECT-TYPE aircraft HAS
    ATTRIBUTES
        ser#:    INTEGER,
        mod#:    STRING;

    SUPERTYPES
        vehicle;

    SUBTYPES
        bombers,
        utility,
        fighters,
        transport,
        fxd_wing,
        rotary.
END-OBJECT-TYPE
```

AIR FORCE DATABASE KS:

```
OBJECT-TYPE transport HAS
    ATTRIBUTES
        capacity:    REAL,
        speed:   REAL,
        range:   REAL;

    SUPERTYPES
    aircraft.
END-OBJECT-TYPE
```

ARMY DATABASE KS:

```
OBJECT-TYPE fxd_wing HAS
    ATTRIBUTES
        capacity:    REAL;

    SUPERTYPES
        aircraft
END-OBJECT-TYPE
```

NAVY DATABASE KS:

```
OBJECT-TYPE utility HAS
    ATTRIBUTES
        config:  STRING,
        passenger_capacity: INTEGER /* inferred
        attribute
        WITH HEURISTICS
        IF config(X):= "personnel"
        THEN passenger_capacity(X) =
        CMPT_PASS_CAP(X)
        /* CMPT_PASS_CAP is a function that
        causes a read to the database (in the appropriate
        language) based on the mod# of the aircraft
        ELSE IF config(X):= "cargo"
            passenger_capacity(X) = CMPT_CAR-
        PASS_CAP(X)
        /* CMPT_CAR-PASS_CAP is a function that
        causes a read to the database (in the appropriate
        language) based on the mod# of the aircraft
            /* e.g., SELECT capacity
            /*      FROM cargo_pass
            /*      WHERE (mod# = mod#(x))

    SUPERTYPES
        aircraft.
END-OBJECT-TYPE
```

Figure 4.12 The KDM Multiple Knowledge Sources

It is assumed that the action officer desires to exclude bombers, fighters, and rotary aircraft, and this is stated to the InHead interface. By means of a predefined KDM "membership" relationship primitive, the thesaurus knows that Army rotary aircraft carry troops and alerts the user to that fact. (Also note that the thesaurus will have knowledge to help the user with query formulation at the syntactic level as well.)

Next, the attribute type troop.capacity is addressed. Troop.capacity is unknown to the global thesaurus KS, but it is known to the local database KSs. The Army database KS has information (via its data dictionary, assumed and not shown in the KS KDM schema) that the attribute "capacity" of fxd_wing aircraft is specified in terms of troops. It has further knowledge that troops are also personnel. The Army database KS places the synonym "personnel.capacity" onto the blackboard and formulates a retrieval operation to its database. Upon seeing personnel.capacity on the blackboard, the Air Force and Navy KSs can now act, since "capacity" of Air Force transport and "config" of the Navy utility object class have heuristics to derive personnel capacity (as

illustrated in the Navy KS schema). This action points out the way the KSs work together to derive more complete answers.

It is apparent from this example that an "intelligent" system, such as the user model and interface, must have a great deal of domain knowledge gained through domain (or enterprise) analysis. This approach allows that domain knowledge to be represented and incorporated in a natural fashion.

4.4.3 An artillery movement example

For this example, suppose that we have an expert system whose task is to provision 10 M110 Howitzer Weapon Systems for departure to the Middle East in 5 days. This expert system is written to interact with three primary databases: 1) a characteristics database, which describes the physical characteristics of the component parts of weapons systems; 2) a weapons systems database which describes the components of weapons systems; and 3) a logistics database which describes the logistics support required to sustain weapons systems in combat. Two secondary, but related databases are: a personnel database for crew requisitioning, and a ships database for obtaining space on seagoing vessels.

The expert system, which plays the role of the user in this example, has a task-oriented functional view of the problem as follows. Potentially semantic ambiguous terms are denoted in bold-face.

Overall Goal: Provision 10 M110 Howitzer Weapon Systems for departure to the Middle East in 5 days

Subgoals

1.0 Determine Availability of 10 M110 Howitzer Weapon Systems.

- 1.1 Determine the locations of such items, subject to constraint of being within 500 miles of Norfolk, Virginia.
- 1.2 Send requests for items to locations to hold for shipment.

2.0 Determine Availability of Logistics Support Units

- 2.1 Specialize camouflage to **desert** conditions.
- 2.2 Specialize radar to **desert** night vision.
- 2.3 Specialize rations to **high water content** rations.
- 2.4 Specialize clothing to lightweight, chemically resistant.

3.0 Determine Availability of Sealift Capability along the Eastern Seaboard.

- 3.1 Calculate total **weight** and **volume** for each system.
- 3.2 Provision **crews** for each system.
- 3.3 Assign **crews** and weapons to ships.
  - 3.3.1 Notify crews.
  - 3.3.2 Send shipment requisitions to sites holding weapons systems.

We now discuss several of the possible cases in which semantic heterogeneity is manifested in this system.

The first ambiguity occurs in the meaning of the word **miles**. The expert system may be assuming nautical miles while the logistics database might be assuming statute miles. When the logistics database sends its answers to the expert system, the data includes measurement units in its data/knowledge packets. The thesaurus is consulted for any unit translations. If ambiguity persists, the user can be consulted to provide appropriate definitions.

By focusing on Subgoal 1.0, we now look at the cooperative problem-solving and active database aspects of InHead. Suppose that in addition to the above databases, the system had an Army installation database, with attributes such as name, type, and location, and a database of Army units that described a unit's location, its weapons systems,its readiness status, and its deployment status. After the expert system retrieves all of the M110 locations, it begins to determine if these installations satisfy the 500 mile constraint. These locations have been returned by installation name and therefore, must be converted to grid coordinates to compute their distance in statute miles from Norfolk, VA. Thus the expert system places a knowledge packet on the blackboard in the form of <OPERATION,OBJECT,RTN_VALUE>, for example, <SELECT,WPN_SYS.LOCN = "FT BRAGG, NC",GRID>.

Because the blackboard allows KSs to see and understand system goals and subgoals, they can actively contribute to the process. In this case the installation database KS, understanding locations in longitude and latitude, helps by placing <nil,INST.LOCN="FT BRAGG, NC",39°N35°W> on the blackboard. The thesaurus KS, knowing that there are several instances of LOCN (figure 4.13 shows a KDM conceptual model of a portion of the thesaurus KS), invokes a method to translate long/lat to grid coordinates and places <nil,LOCN="FT BRAGG, NC",12344321> on the blackboard, which is then used by the expert system to compute the distance from Ft, Bragg, NC to Norfolk, VA.

Another example of a KS actively contributing to satisfy Subgoal 1.0 is found in the KS for the unit database. Noticing on the blackboard that M110s from Ft. Pickett, VA have been targeted for Middle East deployment, the KS checks that availability of M110s on Ft. Pickett. By querying its database the KS can determine the deployment status and readiness category of M110s on Ft. Pickett. If an M110 is already deployed or unfit for combat, the KS could place that information on the blackboard.

KS heuristics can also make their respective databases appear active. For example, reacting to an instruction to transfer 2 M110s from Ft. Pickett to the Middle East, the KS could alert the expert system that the action will cause the readiness category to fall below a certain threshold, resulting in a condition that violates a stated local database constraint.

The expert system might specify the task, "Provide logistic support for ten M110 howitzer systems with desert camouflage." But the logistics support database has an attribute for camouflage in terms of color combinations, rather than the term **desert**. The global thesaurus *knows* that desert colors are grey and brown, so that the semantic heterogeneity is handled easily. Also, if that information were not in the thesaurus, the expert system would engage a dialog with the user to define the term for the thesaurus.

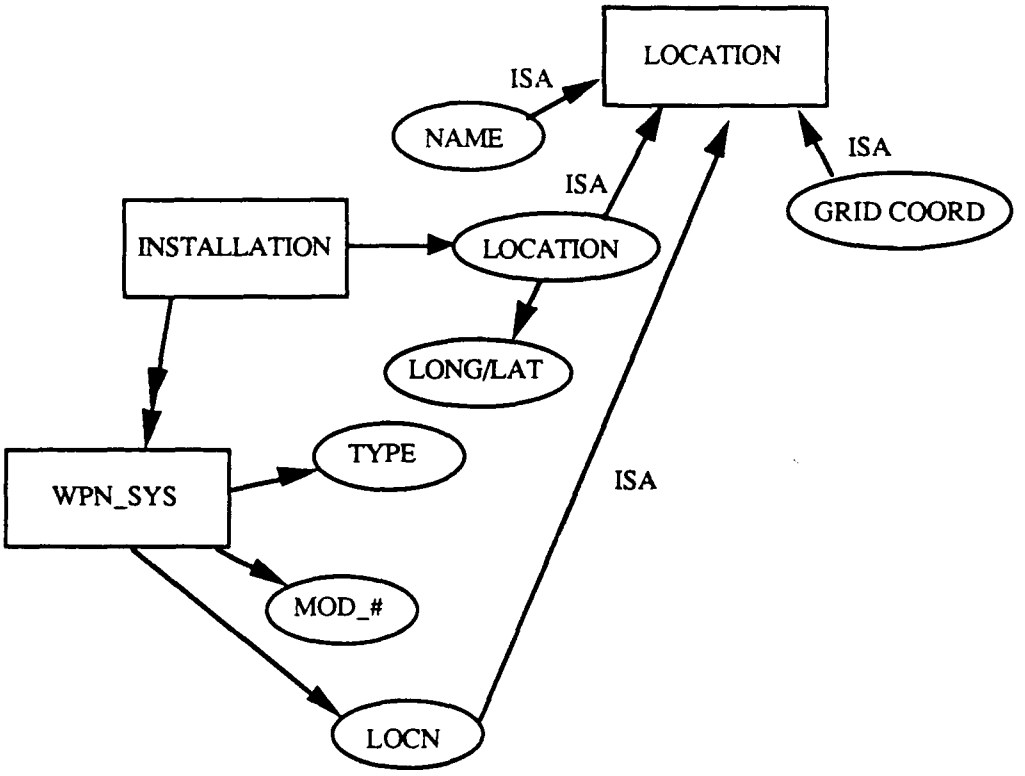


Figure 4.13 Conceptual Model of Several Types of LOCATION

Another instance of ambiguity surrounds the use of the term **crews**. Crews can be either operational crews or maintenance crews. In provisioning crews for each system, the system must know if one or both is needed. That type of information can be placed in the thesaurus. A default rule could be that operational crews take precedence over maintenance crews with the assumption that one maintenance crew can maintain several systems.

5 Coordinated Problem Solving with Multiple Heterogeneous Knowledge Sources

There are two general strategies to problem solving. First, to use a single approach, paradigm, or method, extending its capabilities when necessary due to changes in the problem domain or one's understanding of the domain. This approach has been extensively researched in several disciplines and domains, such as operations research, statistical modeling, simulation, and artificial intelligence (particularly, machine learning).

An alternative strategy to extending or modifying a single approach is to use the results obtained from using that single approach as input to another approach, paradigm, or method. This latter strategy minimizes the complexity of each individual reasoning approach and possibly, too, makes the knowledge representations more "modular." However, the advantages of this strategy are mitigated by the necessarily complicated control of the entire reasoning process. Specifically, the problem of interfacing the various approaches is highly significant (e.g., how to "couple" the approaches, when to use one or the other, and what syntax and semantics to use). This section discusses these problems and suggest possible solutions. Two diverse areas are used to exemplify aspects of the problem; diagnostic problem solving (specifically, automated fault management of telecommunications networks) and database query optimization.

5.1 Multiple knowledge sources for diagnostic problem solving

We view the diagnostic process as beginning when the target system exhibits evidence of a problem; a set of *symptoms* is generated. The diagnostician, or diagnostic system, views these symptoms and searches memories (virtual or physical) where these symptoms have been generated when the target system was in a similar configuration or setting. If an exact match is found, the previously diagnosed cause is proposed as the most likely cause of the current set of symptoms. In other words, *case-based* reasoning is performed. If further testing indicates that this diagnosis is inaccurate, if only a partial match is found (e.g., there were similar symptoms but the setting or the configuration was different), or if this set of symptoms has never been seen before, the diagnostician, or diagnostic system, uses this finding to guide another analysis of the problem. But this analysis uses knowledge of the target system's structure and behavior, together with knowledge of how the target system generates symptoms of the type being observed. In other words, *model-based* reasoning is performed.

The diagnosis from this detailed structural/symptomatic analysis is tested. If it is accurate, a memory of the symptoms, the current target system configuration, and the resulting diagnosis are stored; a new positive *case* is created. If it is not accurate, an explanation of the failure is sought. This explanation may result in some symptoms classified as secondary (i.e., not directly generated by the problem). These secondary symptoms may be removed from further consideration or they may be grouped or clustered when multiple problems are suspected. The diagnostic cycle would be repeated for this same problem resulting in either an accurate diagnosis, or a failure to diagnose the problem with the knowledge currently available. If the former result is achieved, a *positive* case is stored, but if the latter result is achieved, a *negative* case is stored. In this way, the diagnostician or diagnostic system will be able to recognize the limitations of the available knowledge and request additional knowledge, rather than expend resources in what will prove to be a futile effort. An architecture to support such a diagnostic reasoning process is described in Figure 5.1.

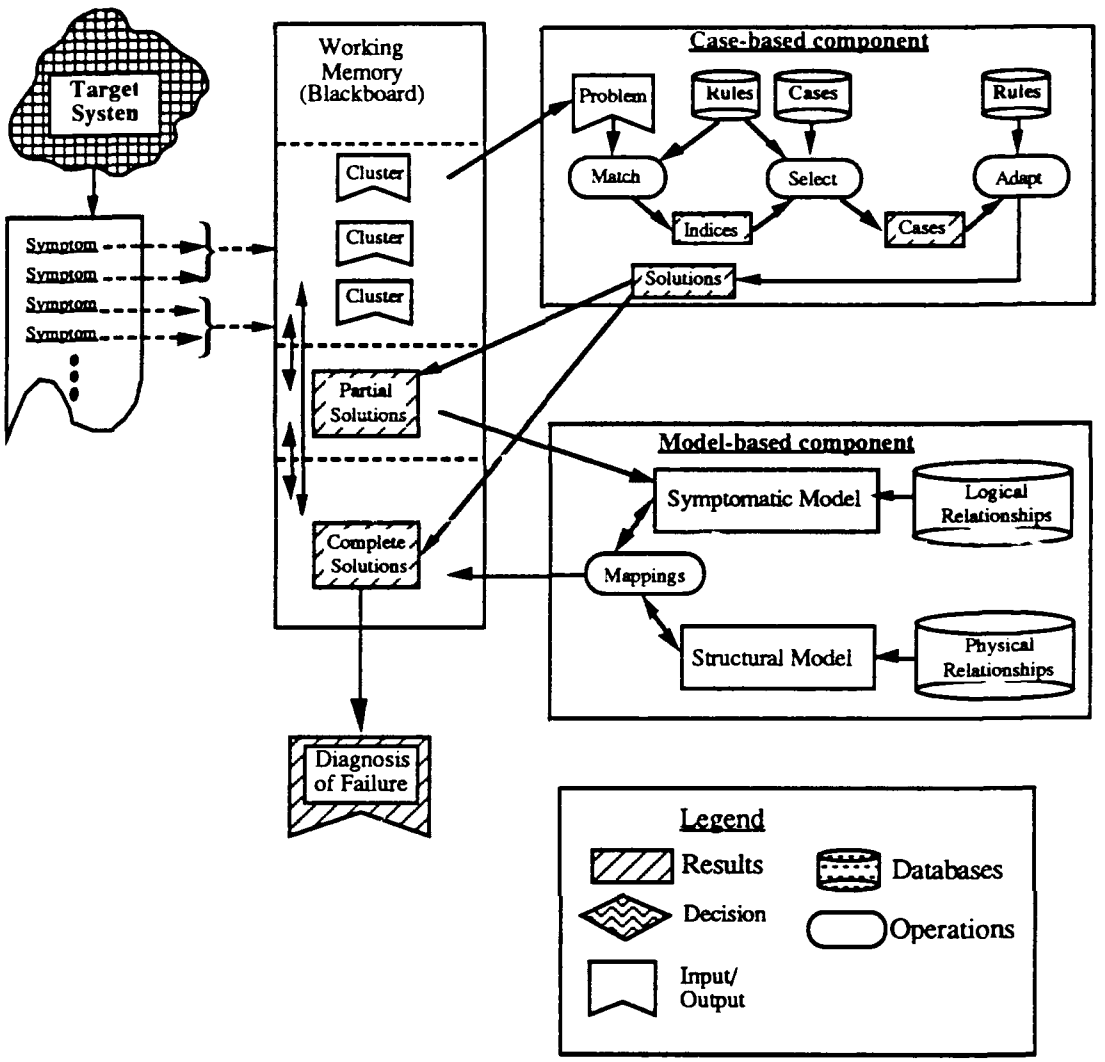


Figure 5.1 Diagnostic System Architecture

It is clear from this description of diagnostic problem solving that heterogeneous information types typically are needed. Additionally, heterogeneous problem-solving strategies are needed. Specifically, a problem-solving approach that reasons with stored problems and their solutions is needed as is a problem-solving approach that reasons with a representation of the structural and symptomatic knowledge. One such possible combination is a case-based reasoning (CBR) system and a model-based reasoning (MBR) system, respectively. To illustrate this, two examples are given; one from the domain of fault management in telecommunications networks and one from the domain of databases, specifically query optimization. Though these two examples may initially seem unrelated, it is shown that the same approach may be applied to both.

5.1.1 Telecommunications fault management example

A complex and evolving domain is telecommunications networks, particularly the transcontinental, long-haul, public switching networks such as those provided by US Sprint, MCI, and AT&T (and all the "Baby Bells" like US West). These networks provide long-distance telephone services by establishing a physical "copper-to-copper" connection between two or more parties through switches, multiplexers, digital cross-connects, and so forth. The "health" of these networks is monitored by technicians known as *surveillance engineers* who monitor situation reports 24-hours a day, seven days a week. These reports are generated by a variety of computer-based systems which are connected to a variety of monitoring devices. As these reports (called *alarms*) scroll across the monitors, the engineers attempt to determine the probable failure and to schedule repair activities. Time is a critical factor in the diagnosis and repair process since the fault may be blocking revenue-generating message traffic and a single blocked or disconnected call can be sufficient cause for a customer to change carriers. But the surveillance engineers may not have all the necessary or relevant information. For example, a piece of equipment can generate an alarm as soon as it is installed. But a surveillance engineer may not be aware of this connection until the physical model of the network is updated. Thus, the surveillance engineer would be seeing what appears to be *phantom* alarm messages; alarms being generated by what appears to be nonexistent equipment.

A main problem, then, in automating failure diagnosis in a complex and evolving system is to capture knowledge about the current physical condition, or *state*, of the system. There are three parts to this knowledge. One part deals with the physical representation of the system; what device is connected to what other device. Another deals with the problem being reported; what devices are reporting the problem. The third deals with the relationship between the first and second parts; under what conditions is problem A reported by device B (or an A-type problem reported by a type B device). The utility of any automated system will depend on the efficiency of knowledge representation, not only in terms of how completely the knowledge is represented but also in how rapidly such knowledge can be used.

A simplified example of a telecommunications network is presented in Figure 5.2 [KBD+90]. This example will be used throughout this section to illustrate how faults arise, how they are detected, and how their cause (or causes) are pin-pointed by network surveillance engineers. Though current telecommunications networks are far more complicated than depicted here, the essential elements are included. The relationship of a general Network Information System (NIS) — the system typically used to collect and monitor information about the network as a whole — is shown by the lines of message flow from entities like switches, digital multiplexers, and so forth. The information flows between the communications network and the NIS are not pursued here, since we are interested in analyzing alarm data, not with how this data arrives.

The data for the example network of Figure 5.2 is described in Table 5.1 [KBD+90] (below) and will be used throughout this section.

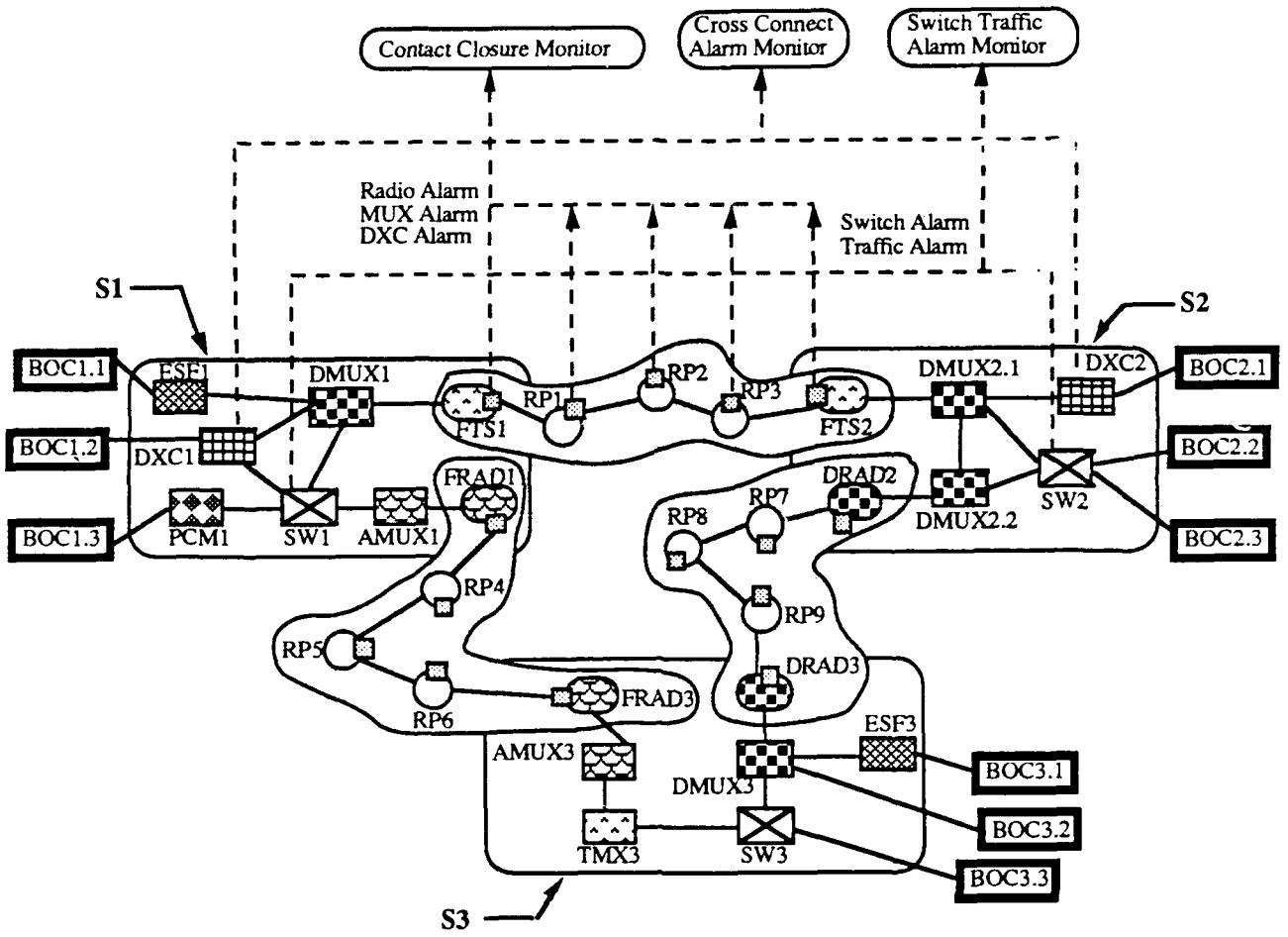


Figure 5.2 Architecture of a Telecommunications Network - An Example

Table 5.1 Sample Data of Example Telecommunications Network

Network Components	Description
AMUX1, AMUX3	Analog Multiplexers (FDM)
Circuit 1, Circuit 2	Circuits
City A, City B	Cities
DMUX1, DMUX2.1, etc...	Digital Multiplexers (TDM)
DRAD2, DRAD3	Digital Radios
DXC1, DXC2	Digital Cross Connects
ESF1, ESF3	Extended Superframe Converters
FRAD1, FRAD3	Fiber Radios
FTS1, FTS2	Fiber Transmission Systems
J1, J2, J3	Junctions
PCM1	Pulse Code Modulation
RP1, RP2, RP3, etc.	Repeaters
S1, S2, S3, S4, S5, S6	Stations
SW1, SW2, SW3	Switches
TMX3	Transmultiplexer (FDM/TDM conversion)
EC1, EC2, EC3	ElementClusters
CCM-1, CCM-2, etc.	ContactClosureMonitors

This knowledge has been expressed in terms of two additional models, both based on the Knowledge Entity-Relationship (KER) model [KBH89]; one to represent the structural knowledge, the other to represent the symptomatic knowledge (the alarms). These models are depicted in Figure 5.3 and Figure 5.4, respectively. The KER Model is based on the concepts and primitives of the Knowledge Data Model presented earlier. The KER also provides for Entity Types and Relationship Types as in the ERA model with the additional semantics that a relationship type may also serve as an entity type as for example the Adjacent Objects, Path, Element Cluster types in Figure 5.3. Moreover, heuristics may be associated with attributes as well as entity and relationship types as in the KDM.

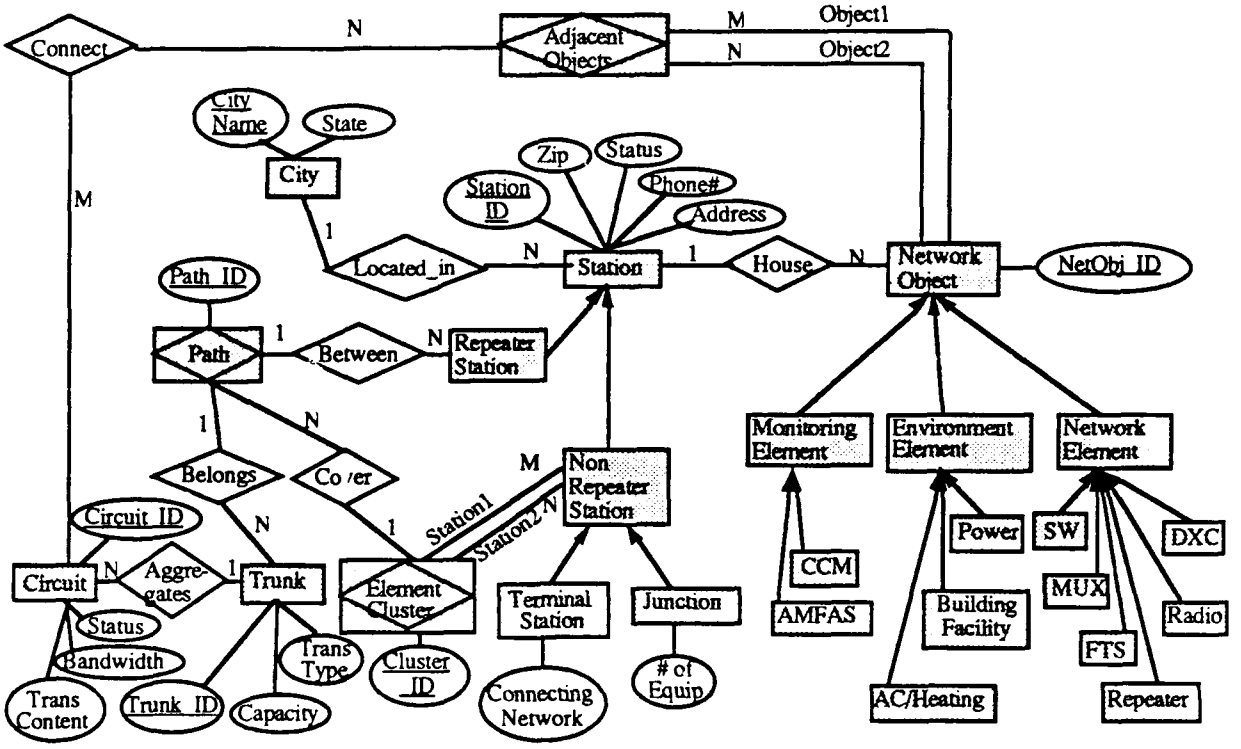


Figure 5.3 KER Structural Diagram of a Telecommunications Network

A problem becomes conspicuous when alarm messages are generated in the Network Information System (NIS). Typically, this information is being continually updated; about every 30 seconds [GW88]. Analysts in the NIS determine the type of problem by using their knowledge about the type(s) of alarms being generated (i.e., symptomatic knowledge) and their knowledge of the network elements which are generating the alarms (i.e., structural knowledge).

Suppose a number of primary faults occur in our example network. Specifically, RP1 drops one of its circuits, radio FRAD1 has a failure, and there is a fiber outage at DMUX1. The following alarms might be simultaneously generated:

- A1 (congestion at radio FTS1)
- A2 (fiber outage at DMUX1)
- A5 (failure at radio FRAD1)
- A3, A4, A6 (congestion at RP1, RP2, and RP3)

A surveillance engineer would notice that all of these alarms are network element alarms (using his/her knowledge of the network's alarm behavior). In other words, each of these alarms is an indicator that at least one fault exists with a device that carries tariff messages. The engineer would use his/her knowledge of the physical network to identify the potentially faulty network elements. This is done by associating the alarms with paths. For example, alarms A1, A3, A4, A6 are generated by elements on same path (cf. Figure 5.2). Since A2 and A5 are on different paths, they are analyzed separately. Finally, the RP1 fault is identified as primary by queries to the monitors of the repeaters in that path (i.e., "If RPx.input=not_okay or RPx.output=not_okay then RPx.status=faulty").

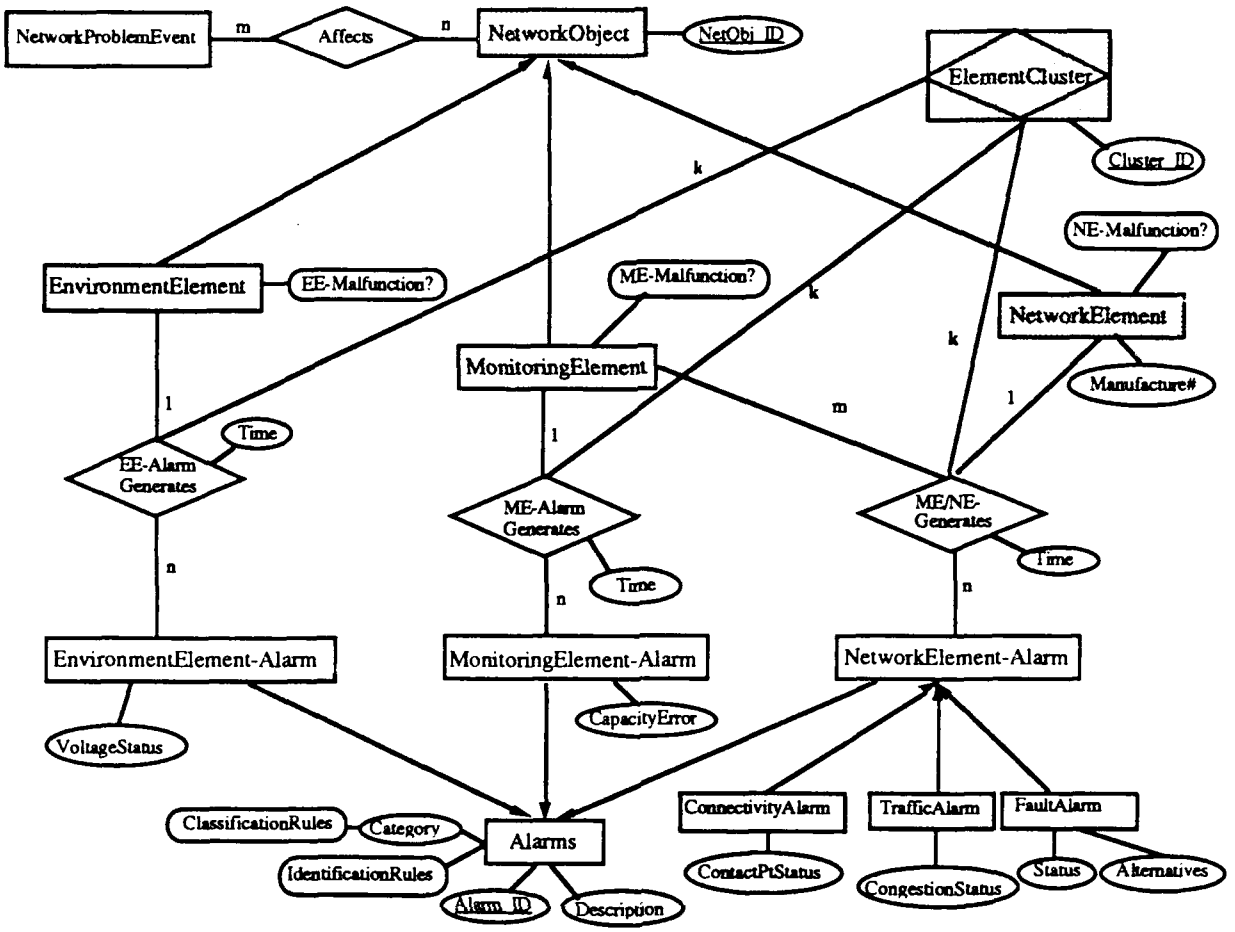


Figure 5.4 KER Diagram of an Alarm Monitoring Network

The first time this particular type of fault occurs, there would be few worthwhile recollections (i.e., either instances of similar faults in different equipment, or different faults in similar equipment, would be recalled). Thus, the recollection step would be quite short and not very useful. A detailed analysis would use knowledge of how the specific switches and repeater are connected along with general knowledge of how switches and repeaters operate. Eventually, the offending repeater would be identified, the switches absolved from blame, and a repair determined. This repair would be implemented and the results retained as a new case.

Usually, the repair/ implementation/ assessment cycle is continued until the problem is fixed. At least the final repair is retained for historical reference (usually informally; like, in the head of the diagnostician). So the next time the same or similar problem is encountered, the detailed diagnostic process is either unnecessary or is streamlined.

5.1.2 Query optimization

A subset of a database is retrieved or updated by a database management system (DBMS) according to criteria, or in response to a set of commands, specified by a query. Typically, the DBMS translates the query into a sequence of basic database operations, specifically, sequences of SELECT, PROJECT, PRODUCT, UNION, DIFFERENCE, JOIN, INTERSECTION, and DIVIDE operations for a DBMS supporting the relational model of data. Each of these sequences

is called a *plan*. It is frequently possible for a query to be decomposable into more than one plan. Query optimization is the process of determining which plan uses a minimum of specified resources, yet accomplishes the tasks specified by that query. This process is vital to the transparent integration of distributed databases, efficient use of very large databases, and rapid response to queries.

There are two stages to query optimization; transforming the queries into strategies (plans), then calculating the cost of these strategies (plans) [JK84]. Algebraic, heuristic, logical, and semantic transformations have been suggested for single and multiple query operations with the greatest amount of interest currently focused on semantic query transformations. Calculating and comparing plan costs is done with a cost model which may be deterministic or based on heuristics. One interesting result of this research is that no single cost model has yet been developed that is consistently better than all other cost models.

One general observation from query optimization research is that accurate database statistics greatly improve the accuracy of the cost model [Chr84]. Of particular significance are query selectivity estimates (also referred to as column cardinality estimates; the number of tuples likely to be retrieved during a SELECT operation). The physical distribution of tuples for a given attribute, and whether or not an index has been created for that attribute, can also influence the performance of the plan. However, as the database is updated, tuple locations and values will change and will invalidate some of these statistics. Only for small, centralized databases would it be feasible to continually recalculate these statistics, even though not doing so will surely degrade the accuracy of the cost model.

An alternative approach is to decrease the cost model's reliance on the specific statistical values as these values become suspect and to incorporate an heuristic evaluation. This approach would still transform the queries into basic database operations, but the first step in forming and optimizing the plan would be to consider adapting previous plans which have performed well (i.e., at low processing costs). These retained plans would have previously been calculated to be optimal, based on the statistical profile and other characteristics of the database. For example, suppose five of the relations in a telecommunications database are as follows, where the doubly-underlined attributes represent keys for their respective relations:

NETWORK_OBJECT(NETOBJ_ID, STATION_ID, OBJECT_TYPE, STATUS)
ALARM(ALARM_ID, NETOBJ_ID, CLUSTER_ID, DESCRIPTION, TIME)
STATION(STATION_ID, PHONE#, ADDRESS, CITY, STATE, ZIP, STATUS)
ELEMENT_CLUSTER(CLUSTER_ID, PATH_ID, STATION_ID)
PATH(PATH_ID, NETOBJ_ID)

For this example, suppose there are the following number of tuples:

500 in NETWORK_OBJECT
2000 in ALARM, 5 of which are DESCRIPTION="Fiber Radio Failure"[1]
10 in STATION, one of which is in Virginia
10 in ELEMENT_CLUSTER
20 in PATH

---

1 We assume that the query optimizer knows this fact.

Now, suppose we are interested in the answer to the question "Which stations were affected when any station in Virginia experienced a *Fiber Radio Failure* alarm?" In the process of answering this query, additional knowledge is gained. For example, that if the relation ALARM is to be used, the first operation should be a SELECT. And if a query refers to the state of Virginia, that adding a SELECT operation on STATION, projecting the single value for STATION_ID, and using it in subsequent SELECT/JOIN operations is likely to be quite efficient. This meta-knowledge could be used in future query optimizations, provided, that is, that the system would know when and how to use this information.

5.2 The role of active data knowledge dictionary/intelligent thesaurus in problem solving

There are at least two roles an active data knowledge dictionary/intelligent thesaurus (ADKD/IT) in problem solving information interchange. First, as an interface between multiple reasoning paradigms. Second, as a means of managing the evolution of knowledge.

5.2.1 As the interface between reasoning paradigms

Should the multiple reasoning paradigms be constrained to a single, globally recognized syntax and semantics? This depends on how tightly coupled they are or need to be. If tightly coupled, then such a single syntax and semantics must be used. However, this eliminates a benefit of using multiple approaches; allowing each approach to use the knowledge representation best suited to the particular paradigm (e.g., Memory Organization Packages, MOPs, for cases and rules for heuristics). A potential disadvantage of loosely coupling the multiple reasoning paradigms is that it allows the possibility of losing semantic integrity. In other words, transferring complex data or knowledge from a system with a rich semantics to one with more rudimentary semantics could result in some crucial interpretation being lost. An ADKD/IT could mitigate this possibility by capturing and maintaining the nuances of representation methods and acting as a translator between the paradigms. A bonus would be that since the semantic and syntactic knowledge would be explicitly controlled by a single knowledge source, evolving the knowledge is more straight forward. This is particularly important when using historical knowledge.

5.2.2 To manage knowledge evolution

Historical knowledge is not totally obsolete when the target system evolves. Neither is it completely accurate. An ADKD/IT could support historical knowledge evolution by isolating the relationships between the cases and the essential features of the network's structural and behavioral knowledge. In other words, to maintain the *context* of the cases. We are currently researching possible canonical forms for this historical knowledge. The forms presented by Ullman in discussing query optimization algorithms look promising [Ull89]. We are also considering the forms suggested by Shank in [RS89]; *ossified cases*, *paradigmatic cases*, and *stories*. We have not, however, found these latter representations to be readily generalizable. We are continuing our research in these areas and feel that the multiple knowledge source, cooperative problem-solving approach to large scale heterogeneous systems is an appropriate one.

6 Dictionary support for schema evolution in object-oriented databases

In conducting research on an Active Data/Knowledge Dictionary (ADKD), one area of interest is how such a dictionary could support the users' efforts to manage database schema evolution. Data dictionaries are not static repositories of information. Rather, the metadata stored in a data dictionary is constantly changing as users design databases to support their applications and then as they modify those designs to reflect the changing requirements of their applications.

In this section, we discuss data dictionary support for schema evolution specifically in the context of object-oriented databases. We begin with a review of major concepts common to many object-oriented data models, and from these concepts we will establish working definitions for terms such as schemata and metadata as they apply to object-oriented databases. Then we discuss the problems associated with schema evolution, illustrating them with an example session with the GemStone object-oriented DBMS. Next we review literature on schema evolution for proposed solutions to these problems. After critiquing these previously proposed solutions, we present our own proposal for a data dictionary that is capable of reasoning about evolving schemata, and finally we describe further research that needs to be conducted if such a data dictionary is to become a reality.

6.1 Object-oriented data models

Before we can discuss data dictionary support for schema evolution in the context of object-oriented databases, we must first review the major concepts common to many object-oriented data models. From these concepts we will arrive at working definitions for terms such as schemata and metadata as they apply to object-oriented database management systems.

The lack of a single reference model. It is important to note that there is not a single, accepted object-oriented data model. Kim states in *Research Directions in Object-oriented Database Systems* that there is "no standard for object-oriented databases" and that there is "no standard object-oriented database language in which to program applications" [Kim90, page 1]. This problem exists because there is no single, precisely-defined object-oriented data model. The Object-Oriented Databases Task Group (OODBTG) of the Database Systems Study Group (DBSSG)[1] is addressing this problem, and their latest draft reference model for object data management identifies a set of characteristics commonly-shared among object-oriented database management systems.

[1] One of the advisory groups to the Accredited Standards Committee X3 (ASC/X3), Standards Planning and Requirements Committee (SPARC), operating under the procedures of the American National Standards Institute (ANSI).

Commonly-shared characteristics. The itemized lists below present the commonly-shared characteristics of object-oriented database management systems as identified by the OODBTG in [Obj90].

- Abstract Object Characteristics
  - Encapsulation
  - Objects
  - Identity
  - Types and Classes
  - Composition
  - Polymorphism
  - Inheritance
  - Extensibility
  - Information Semantics
- Data Management Characteristics
  - Persistence
  - Data Language
  - Integrity Constraints
  - Concurrency
  - Transactions
  - Recovery
  - Versions
  - Distribution
  - Security
  - User Interfaces

It is not within the scope of our report to provide detailed descriptions of each of these characteristics — reference [Obj90] takes fifteen pages to do so. Suffice it to say that we have considered each of these characteristics in coming up with the following working definitions of terms that will be used throughout the remainder of our discussion.

Working definitions. The definitions in figure 6.1 are provided for readers who may be unfamiliar with object-oriented database systems and the terminology used to describe them. Terms used in the body of a definition which will be further defined are set in boldface type. Other important terms appearing in the body of a definition are set in italics.

6.2 Problems posed by evolving schemata

We stated at the beginning of section 6 that data dictionaries are not static repositories of information. Rather, the metadata stored in a data dictionary is constantly changing as users design databases to support their applications and then as they modify those designs to reflect the changing requirements of their applications. This continuing design activity poses two problems for users of object-oriented database systems, as enumerated below.

1. As a schema evolves, inconsistencies arise between information inferred from the metadata and information stored persistently by the database system. Appendix B provides an example session with the GemStone object-oriented database system which illustrates this fact.

2. As the users' understanding of the problem to be solved changes, their search for a new solution may lead them to re-derive metadata associated with previous solutions because object-oriented databases do not support *versions* of either classes or schemata.

Object-oriented database system: a database system which employs an **object-oriented language** for data definition, manipulation, and query formulation.

Object-oriented language: a language which "provides linguistic support for **objects**" and which supports the "management of collections of objects by means of **classes** and class **inheritance**" [Weg87b, pages 508–509].

Objects: objects have "a set of operations and a state that remembers the effect of the operations. Objects communicate by sending each other messages to perform operations. Objects may be contrasted to functions which have no memory. Whereas function values are completely determined by their arguments, an object may learn from experience, its reaction to an operation being determined by its invocation history" [Weg87b, page 508].

Classes: every object in an object-oriented database is an instance of some class, alternatively known as a *type*, which specifies behavior for its instances through the following three constructs:

- a set of *operations*, alternatively known as *methods* or *routines*, defined for all instances of the class;
- a set of *properties*, alternatively known as *attributes* or *instance variables*, similarly defined for all instances of the class; and
- a set of *constraints*, alternatively known as *invariants* or *assertions*, which must be satisfied by all instances of the class at all times.

Inheritance: a relationship that may exist between two classes, referred to as a *subclass* and its *superclass*. The inheritance relationship specifies that all operations, properties, and constraints defined for a superclass will be defined for its subclass. A system which supports *multiple inheritance* permits classes to have more than one superclass from which they inherit operations, properties, and constraints.

Schema: in the context of an object-oriented database system, the term schema refers to a set of classes organized into a **class hierarchy** if multiple inheritance is not supported or into a **class lattice** if multiple inheritance is supported. The definitions of these classes are stored in a *data dictionary* and are collectively referred to as the *metadata*.

Class hierarchy: a strict hierarchy in which each node represents a class and in which the parent-child relationship of the hierarchy represents the inheritance relationship between a superclass and its subclass.

Class lattice: a "directed acyclic graph with exactly one node containing no outgoing arcs" [SZ87, page 395]. Each node in the graph represents a class, and each arc in the graph represents an inheritance relationship, pointing from a subclass to its superclass.

Figure 6.1: Object-oriented database system terminology

The first problem cited above affects the quality of solutions derived; the second affects the efficiency with which solutions can be derived. During the past five years, various researchers have proposed solutions to these problems associated with schema evolution. We review and critique their proposed solutions in the next section of this paper before characterizing our proposed solution in section 6.4.

6.3 Review of schema evolution literature

Object-oriented databases have only recently emerged as a topic of research and development, and as a consequence, many research areas associated with object-oriented databases have not been investigated in depth. One such area is that of schema evolution. Kim states in *Research Directions in Object-oriented Databases* that works published to date establish a "framework for the evolution of an object-oriented database schema [which] represents important first steps towards the logical design of databases," but that no object-oriented database system yet has implemented *versions* of either individual classes or entire schemata [Kim90, page 13]. In the sections that follow, we will review the published works of those groups whose research has helped establish this framework for schema evolution in object-oriented database systems.

6.3.1 Servio Logic Corporation (GemStone)

Papers published by researchers associated with the GemStone object-oriented database management system describe a methodology for handling class and schema modifications in GemStone [B+89, PS87]. Their methodology begins with the identification of six *invariants of schema modification*, two of which are presented in figure 6.2.

To ensure that these invariant properties of a schema are maintained as classes are modified, the methodology next identifies eight primitive *class-modifying operations*, one of which is presented in figure 6.3.

By identifying invariants of schema modification and describing primitive class-modifying operations which maintain those invariants, the methodology put forth by Servio researchers guarantees that modifications made to a consistent schema will result in another consistent schema; however, it does not support versions of either classes or schemata, and therefore only deals with the first problem cited in section 6.2.

It is important to note that while Servio researchers have described a method for coping with change at the schema level, that method has not been implemented in the commercial offering of GemStone. The example session with GemStone presented in appendix B demonstrates that changes made to a class are not propagated by the system, and that as a result, the schema resides in an inconsistent state with its invariant properties unsatisfied.

Representation invariant "The representation of an object is determined by its class. An object's storage format, size, variables, and constraints upon those variables must be as specified by the object's class" [B+89, page 300].

Full inheritance invariant "The representation of instances determined by a class is inherited by all of its subclasses. ... Every class inherits every instance variable defined in its superclass" [B+89, page 301].

Figure 6.2: Two GemStone invariants of schema evolution

Adding a named instance variable "All instances of a class may have an additional instance variable defined. In order to preserve the representation invariant, adding a named instance variable is not allowed if another instance variable already has the same name. ... If the named instance variable is not defined in a subclass, then the modification is propagated to that subclass. This preserves the full inheritance constraint. All instances of the class to which the instance variable has been added gain a value of `nil` for the new instance variable in order to maintain the representation invariant" [PS87, pages 113–114].

Figure 6.3: A GemStone class-modifying operation

6.3.2 Brown University

Brown University researchers Skarra and Zdonik have also actively investigated schema evolution in the context of object-oriented database systems [SZ86, SZ87]. Their work addressed the problem described below.

> "In this work we look at the problem of changes in type definitions (i.e., the database schema). It is natural to assume that in the course of a design, views of the world (i.e., type definitions) will change. A database stores objects for long periods of time. Each object was created as an instance of some type at some point in that type's evolution. The type described all of the assumptions about that object's behavior. What happens when that type definition changes? Old objects may be incompatible with a new definition of a type. Also, new objects may be incompatible with old definitions of their type for which programs have already been written" [SZ87, pages 393–394].

The solution proposed by Skarra and Zdonik involves maintaining *versions* of types (classes) and associating *exception-handling mechanisms* with each version so that all instances of all versions of a type behave uniformly. Any time a change is made to a type T, a new version of that type is created. Associated with each type is a *version set interface* which contains "all the properties and operations ever defined by some version of T and every value ever declared valid for properties and operation parameters" [SZ87, page 402].

All instances of all versions of a type are expected to behave according to the type's version set interface. So that they do, each version of a type has associated with it exception handling mechanisms in the form of *prehandlers* and *posthandlers*. These prehandlers and posthandlers work as shown in figure 6.4 [SZ87, page 403]. When a program *P(t)* written under type definition *T3* needs to access or update an instance of T that was created under version *T1*, either a prehandler associated with *T1* or a posthandler associated with *T3* will execute if an exception is raised during the operation [SZ87].

By maintaining versions of types and associating exception handling mechanisms with each version, Skarra and Zdonik hope to avoid converting the instances of a type when a new version of that type is created. It should be noted, however, that their proposed solution depends on the fact that meaningful exception handling mechanisms can be defined, something which will not always be possible. Also, their proposed solution does not provide for identifiable versions of schemata. Since their proposed solution groups all versions of a type into a single version set interface, only one schema can be supported at any given time.

6.3.3 MCC (ORION)

Papers published by researchers at the Microelectronics and Computer Technology Corporation (MCC) in the past five years describe their prototype object-oriented database system, ORION, and its approach to solving the problems of schema evolution [B+87a, B+87b, KC88, K+87, K+89]. The paragraphs below review their works which address the two problems of schema evolution identified in section 6.2: ensuring consistency within a schema and providing for versions of metadata.

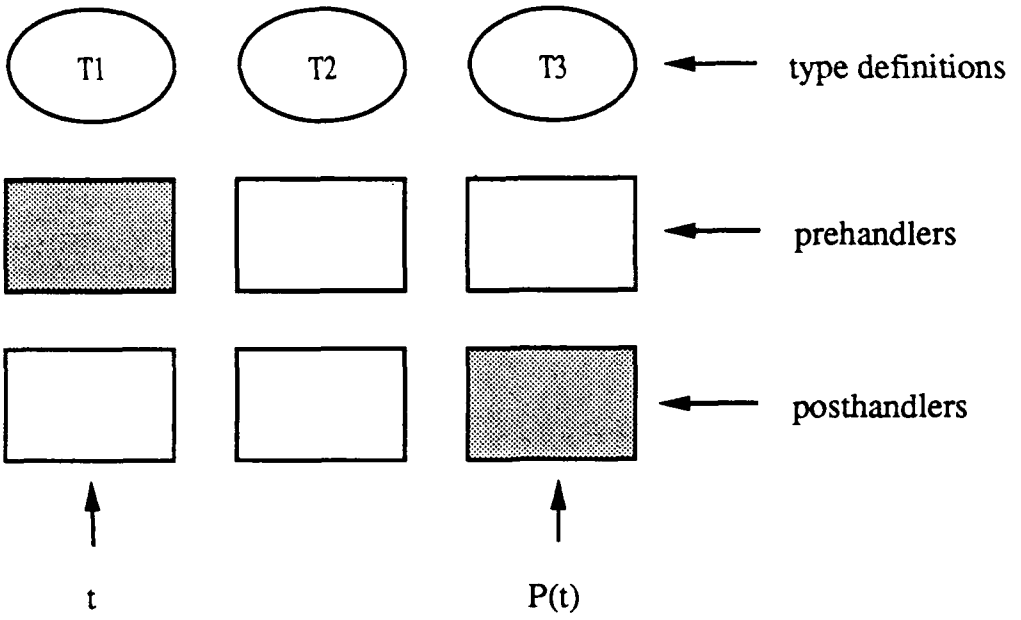


Figure 6.4: Skarra and Zdonik's exception handling mechanisms

Full inheritance invariant "A class inherits all instance variables and methods from each of its superclasses, except when full inheritance causes a violation of the distinct name and distinct identity invariants. In other words, if two instance variables have distinct origin but the same name in two different superclasses, at least one of them must be inherited. If two instance variables have the same origin in two different superclasses, only one of them must be inherited" [B+87b, page 314].

Figure 6.5: An ORION invariant of schema evolution

Ensuring consistency within a schema. The approach adopted by MCC researchers for ensuring consistency within a schema begins in the same fashion as the GemStone methodology described in preceding paragraphs — by identifying the *invariants of schema evolution.* The invariants defined by MCC researchers for ORION are similar in nature to those defined for GemStone, but they differ in detail because the data models underlying these two object-oriented database systems are not the same. For example, the full inheritance invariant defined for ORION, presented in figure 6.5, differs from that defined for GemStone because the ORION data model supports multiple inheritance where the GemStone data model does not.

Where the GemStone methodology identified only eight primitive class modifying operations, the MCC researchers identified a *taxonomy of schema changes* [B+87b, page 316], shown in figure 6.6, containing 21 such possible changes. There are two reasons for this profound difference in the number of possible changes. First, under the ORION data model, a schema is represented as a class lattice which is a more complicated structure than the simple class hierarchy supported by the GemStone data model. Secondly, the eight primitive class

1. Changes to the contents of a node (a class)
   (a) Changes to an instance variable
      i. Add a new instance variable to a class
      ii. Drop an existing instance variable from a class
      iii. Change the name of an instance variable of a class
      iv. Change the domain of an instance variable of a class
      v. Change the inheritance (parent) of an instance variable (inherit another instance variable with the same name)
      vi. Change the default value of an instance variable
      vii. Manipulate the shared value of an instance variable
         A. Add a shared value
         B. Change the shared value
         C. Drop the shared value
      viii. Drop the composite link property of an instance variable
   (b) Changes to a method
      i. Add a new method to a class
      ii. Drop an existing method from a class
      iii. Change the name of a method of a class
      iv. Change to the code of a method of a class
      v. Change the inheritance (parent) of a method (inherit another method with the same name)
2. Changes to an edge
   (a) Make a class S a superclass of a class C
   (b) Remove a class S from the superclass list of a class C
   (c) Change the order of superclasses of a class C
3. Changes to a node
   (a) Add a new class
   (b) Drop an existing class
   (c) Change the name of a class

Figure 6.6: ORION's taxonomy of schema changes

Rule1: "If an instance variable is defined within class C, and its name is the same as that of an instance variable of one of its superclasses, the locally defined instance variable is selected over that of its superclass. The same applies to methods" [B+87b, page 315].

Rule 8: (Edge Removal Rule) "If class A is the only superclass of class B, and A is removed from the superclass list of class B, then B is made an immediate subclass of each of A's superclasses. The ordering of these new superclasses of B is the same as the ordering of the superclasses of A.

A corollary to Rule 8 is that, if the root class OBJECT is the only superclass of a class B, any attempt to remove the edge from OBJECT to B is rejected. If the edge is removed, node B would become isolated, since OBJECT has no superclass to which B may be linked as a new superclass" [B+87b, page 316].

Figure 6.7: Two ORION rules of schema evolution

modifying operations defined for GemStone are not complete in a graph theoretic sense as they do not account for explicit deletion of a class from the hierarchy [PS87].

Associated with each of the 21 schema changes possible under the ORION data model are a number of rules which serve to maintain the schema invariants. Two of these rules are presented in figure 6.7, and a complete list of the 12 *rules of schema evolution* is presented in [B+87b].

By identifying a taxonomy of schema changes, the invariants of schema evolution, and a set of rules which maintain those invariants as changes are made, the methodology put forth by MCC researchers ensures that changes made to a consistent schema will result in another consistent schema. Later work published by this group addressed the problem of versioning metadata as described in the following section.

Providing for versions of metadata. As noted earlier in this paper, no object-oriented database system has yet implemented versions of either classes or schemata. To the best of our knowledge, the only proposal for versioning metadata has been presented by MCC researchers Kim and Chou in [KC88]. This proposal defines a model of versions of schemata in terms of seven rules, the first three of which are presented in figure 6.8.

The three rules in figure 6.8 define the *capabilities* of schema versions and their relationships to one another in the *version-derivation hierarchy*. The remaining four rules of Kim and Chou's model of versions of schemata define the *access scope*, or set of objects accessible to a schema version.

It is important to keep in mind that this proposal addresses only versions of schemata; it does not support versions of classes. Kim and Chou note that an alternative approach to versioning schemata would be to treat classes as versionable objects from which 'virtual' schema versions could be constructed.

Schema-Version Capability Rule: "A schema version may be either a transient schema version or a working schema version. A transient schema version may be updated or deleted; and it may be promoted to a working schema version at any time. A working schema version cannot be updated. A working schema version may be deleted or demoted to a transient version, if it has no child schema version" [KC88, page 151].

Schema-Version Derivation Rule: "Any number of new versions of schema may be derived at any time from any existing schema version, giving rise to a version-derivation hierarchy for the schema. A derived schema version is initially a transient schema version. If a schema version is derived from a transient schema version, the transient schema version is automatically promoted to a working schema version" [KC88, page 151].

Schema-Version Deletion Rule: "A schema version which is a leaf node in the schema-version derivation hierarchy can be deleted, regardless of whether it is a working version or a transient version. A schema version cannot be deleted, if it has any child schema version. When a schema version is deleted, its direct access scope is also deleted" [KC88, page 151].

Figure 6.8: Three schema-version rules

6.3.4 Other related research

Besides the works reviewed in sections 6.3.1 through 6.3.3. little else has been published on the subject of schema evolution in object-oriented database systems. Björnerstedt and Hultén have adopted a type version scheme for their object management system, AVANCE, which is quite similar to the solution proposed by Skarra and Zdonik [*BH89*]. Li and McLeod have also investigated type evolution in the context of an object-oriented data model, but they emphasize the use of machine learning techniques to (semi-)automatically evolve type definitions. Their work does not support versions of either classes or schemata [LM88b, LM88a].

While there has not been much published on the subject of schema evolution in object-oriented database systems, there has been much published on version control in object-oriented database systems [BM88a, CK86, CK88, KC87, K+86]. Some of these works [CK86, CK88, K+86] provide the basis for Kim and Chou's proposal for versions of schemata, but none specifically address the problem of versioning metadata.

6.3.5 Critique

In sections 6.3.1 through 6.3.3 we reviewed the published works of three groups who have proposed solutions to problems associated with schema evolution in object-oriented databases. We believe that the problem of ensuring consistency within a schema can be adequately addressed by an approach similar to that proposed for the ORION database system. We find, however, that none of these proposed solutions satisfactorily provide for versioning metadata.

- Servio researchers include no provision for versioning metadata in their published works on schema evolution.

- Skarra and Zdonik's proposal provides for versions of types (classes), but groups all these versions into a single version set interface which can support only one schema.
- MCC researchers proposed versioning entire schemata, but not individual classes.

Since classes are the basic unit to which changes are made and from which schemata are constructed, we feel that any approach for versioning metadata should provide for identifiable versions of classes. As noted previously by Kim and Chou, virtual schema versions can be constructed from versionable classes, and in the next section we propose a mechanism for doing just this.

6.4 Proposed solution

Our proposed solution to the problem of versioning metadata comes from the observation that a data dictionary must recognize and maintain the dependencies among differing versions of classes[2] if it is to construct virtual schema versions from those classes. A mechanism which has been developed for maintaining knowledge of dependencies among data items is the *truth maintenance system* (TMS) [Doy79, dK86a, dK86b, dKF90]. We propose giving the data dictionary associated with an object-oriented database system the reasoning capabilities of an assumption-based TMS (ATMS)[3] so that it may construct consistent schemata from a collection of versioned classes. In the sections to follow, we describe TMS support for problem solving in general, and then describe how an ATMS may be used when the problem solving task at hand is database design.

6.4.1 TMS support for problem solving

Truth maintenance systems can be used as part of an overall architecture for problem solving as shown in figure 6.9, a description of which follows.

> "The problem solver architecture reflects [a] natural partitioning of concerns. The inference engine is concerned with making inferences within the problem solving task domain; the TMS is concerned with organizing the problem solving process such that the current beliefs and inferences of the inference engine are consistent with each other. Problem solving then proceeds by the continuous interchange of information between the inference engine and the truth maintenance system" [dKF90, page MA5-180].

[2]Dependencies among classes arise from the *client* and/or *descendant* relationships that may exist between them. If a class A has an attribute of type B, then class A is a *client* of class B. If class A inherits from class B, then class A is a *descendant* of class B.

[3]The ATMS is one of several different types of truth maintenance systems identified by Forbus and de Kleer in [dKF90]. Each of these different types of TMSs is well suited to a particular type of problem solving. We have chosen to work with the ATMS because it supports the simultaneous exploration of multiple potential solutions — something that database designers often do when modeling complex application domains.

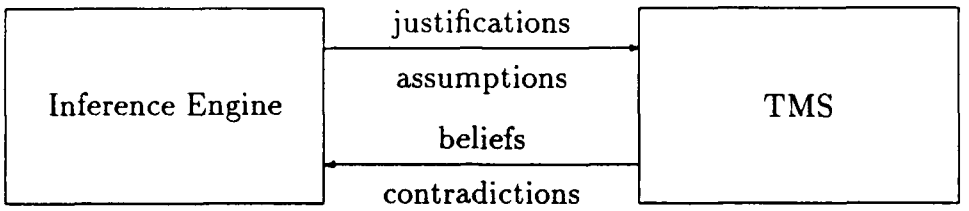


Figure 6.9: Problem-solver architecture

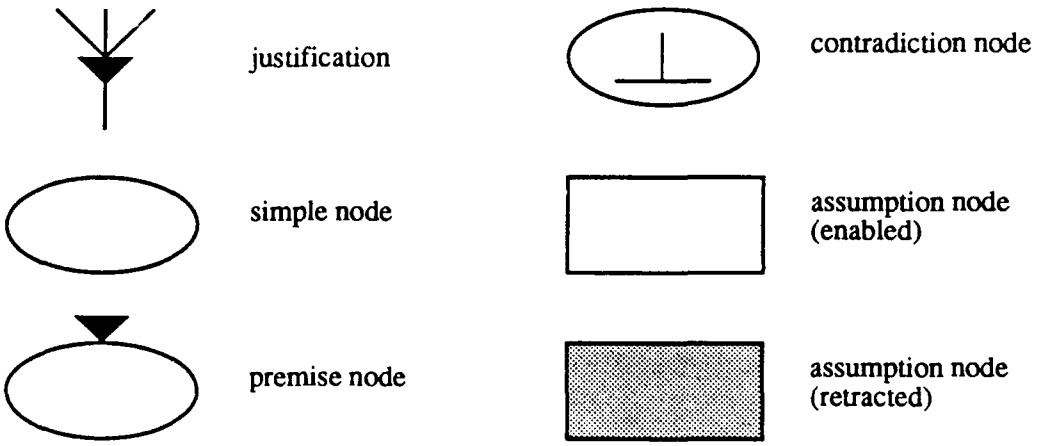


Figure 6.10: Dependency graph symbols

When the problem solving task domain is logical database design, this general architecture can be interpreted as follows. The database designer serves as the inference engine, supplying metadata in the form of class definitions to the data dictionary. The data dictionary, given the reasoning power of a TMS, can then serve as a cache for the metadata, detect inconsistencies in the metadata, identify consistent schemata from the metadata, and construct explanations for its conclusions.

The state of a TMS's knowledge about an ongoing problem solving process can be represented by a *dependency graph* constructed from the symbols shown in figure 6.10. The nodes of a dependency graph represent basic propositions assumed or derived during the process of problem solving. The justifications linking these nodes represent dependencies among the basic propositions.

Each node in a dependency graph will be represented within the ATMS by a data structure of the form shown below [dK86a, page 146].

$$\langle\ \textit{datum, label, justifications}\ \rangle$$

The *datum* in this data structure uniquely identifies a node in a dependency graph. When the ATMS is used to manage schema evolution in an object-oriented database system, the datum will identify a specific version of a class.

The set of *justifications* in the ATMS data structure represents a set of propositional Horn clauses, each of which has the *datum* as its consequent. For example, justifications in the ATMS data structure shown below

$$\langle\ \textit{datum, label, } \{(z_1, z_2, \ldots), (y_1, y_2, \ldots), \ldots\}\rangle$$

represent the following set of propositional Horn clauses.

$$\begin{array}{rcl} (z_1 \wedge z_2 \wedge \ldots) & \rightarrow & \textit{datum} \\ (y_1 \wedge y_2 \wedge \ldots) & \rightarrow & \textit{datum} \\ \ldots & \rightarrow & \textit{datum} \end{array}$$

When the ATMS is used to manage schema evolution in an object-oriented database system, these justifications will represent the dependencies among specific versions of classes as determined by their client and/or descendant relationships.

The *label* in the ATMS data structure contains a minimal, complete set of *environments* where each environment is a consistent set of *assumptions*. This label can also be interpreted logically as a set of propositional Horn clauses. For example, the label in the ATMS data structure shown below

$$\langle\ \textit{datum, } \{\{A_1, A_2, \ldots\}, \{B_1, B_2, \ldots\}, \ldots\}, \textit{ justifications}\ \rangle$$

represents the following set of propositional Horn clauses.

$$\begin{array}{rcl} (A_1 \wedge A_2 \wedge \ldots) & \rightarrow & \textit{datum} \\ (B_1 \wedge B_2 \wedge \ldots) & \rightarrow & \textit{datum} \\ \ldots & \rightarrow & \textit{datum} \end{array}$$

In the context of object-oriented database design, a specific version of a class which has no client or descendant classes will be considered an assumption. Each consistent set of assumptions forming an environment corresponds to a specific schema version. Therefore, by examining the labels associated with the ATMS data structures, it will be possible to readily identify all consistent schemata that can be constructed from a collection of versioned classes.

In order to demonstrate how an ATMS may be used to support database design efforts, the following section provides an example of schema evolution in an object-oriented database. Accompanying the example is a series of figures containing dependency graphs and ATMS data structures which illustrate how the reasoning portion of our data dictionary would keep track of the ongoing process of database design.

6.4.2 An example of schema evolution

Initially, the reasoning portion of our data dictionary will record as *premises* the existence of fundamental classes such as *OBJECT*, *STRING*, and *INTEGER* as shown in figure 6.11.

If the database designer were to define a class A with *OBJECT* as a superclass and with an attribute of type *STRING*, the reasoning portion of our data dictionary would record this first version of class A as an *assumption* as shown in figure 6.12.

If the database designer were to define a class B with A_1 as a superclass and with a function returning a value of type *INTEGER*, the reasoning portion of our data dictionary would record this first version of class B as an assumption as shown in figure 6.13. Note that class A_1 is no longer treated as an assumption because its inclusion in a schema would be justified by the inclusion of class B.

If the database designer were to modify the definition of class A_1 by changing the name of its attribute, the reasoning portion of our data dictionary would record as an assumption the existence of a second version of class A as shown in figure 6.14. Because two versions of the same class cannot co-exist in a consistent schema, the reasoning portion of our data dictionary would also record the fact that classes A_1 and A_2 are inconsistent with one another as shown. It also follows from this dependency graph that classes B_1 and A_2 cannot co-exist within a consistent schema since B_1 justifies the inclusion of A_1 which is inconsistent with A_2.

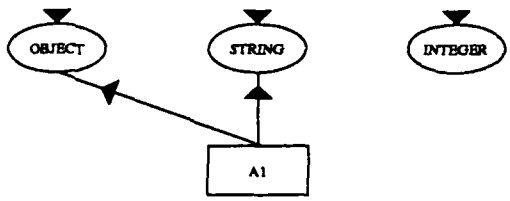
If the database designer wanted a version of class B which was consistent with A_2, it would be necessary to re-compile the definition of class B_1, specifying A_2 as its superclass. With this done, the reasoning portion of our data dictionary would record as an assumption the existence of a second version of class B as shown in figure 6.15. Also recorded would be the fact that the two versions of class B are inconsistent with one another. Note that class A_2 is no longer treated as an assumption because its inclusion in a schema would now be justified by the inclusion of class B_2.

Finally, consider the case where the database designer defines a class C with A_1 as its superclass, and for some reason, this first version of class C violates the schema invariants defined for its data model. The reasoning portion of our data dictionary will record this fact by adding a premise that C_1 is *nogood* and cannot be part of any consistent schema, as shown in figure 6.16.

From the set of versioned classes in figure 6.16 there are three consistent schemata that can be readily identified by examining the labels associated with the ATMS data structures — these three schemata are shown in figure 6.17. The first schema is produced by enabling the assumption node representing class B_1 and retracting all other assumption nodes. It follows from the dependency graph then that this schema would consist of classes A_1, B_1, $OBJECT$, $STRING$, and $INTEGER$. The second schema is produced by enabling the assumption node representing class B_2 and retracting all other assumption nodes. This schema would consist of classes A_2, B_2, $OBJECT$, $STRING$, and $INTEGER$. The third schema is the trivial case in which no assumption nodes are enabled and only the fundamental classes $OBJECT$, $STRING$, and $INTEGER$ will be part of the schema.

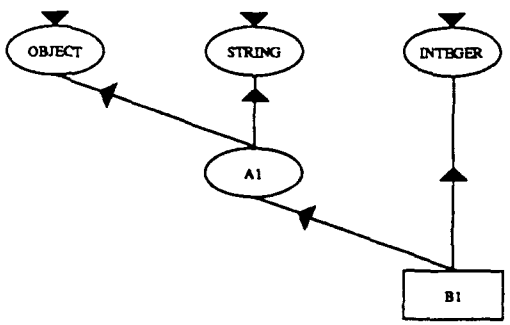
$\langle OBJECT, \{\{\}\}, \{()\}\rangle$
$\langle STRING, \{\{\}\}, \{()\}\rangle$
$\langle INTEGER, \{\{\}\}, \{()\}\rangle$

Figure 6.11: Fundamental classes recorded as *premises*



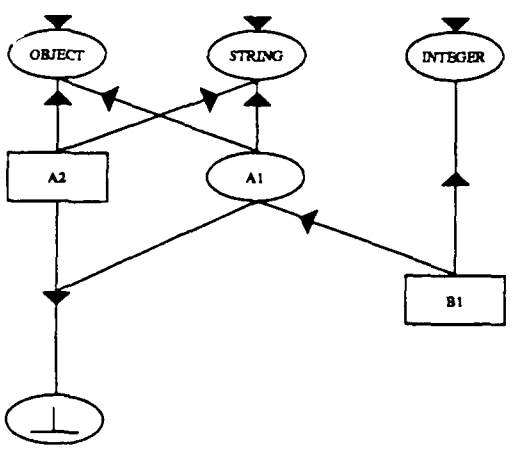
$\langle OBJECT, \{\{\}\}, \{(), (A_1)\}\rangle$
$\langle STRING, \{\{\}\}, \{(), (A_1)\}\rangle$
$\langle INTEGER, \{\{\}\}, \{()\}\rangle$
$\langle A_1, \{\{A_1\}\}, \{(A_1)\}\rangle$

Figure 6.12: Newly-defined class A recorded as an *assumption*



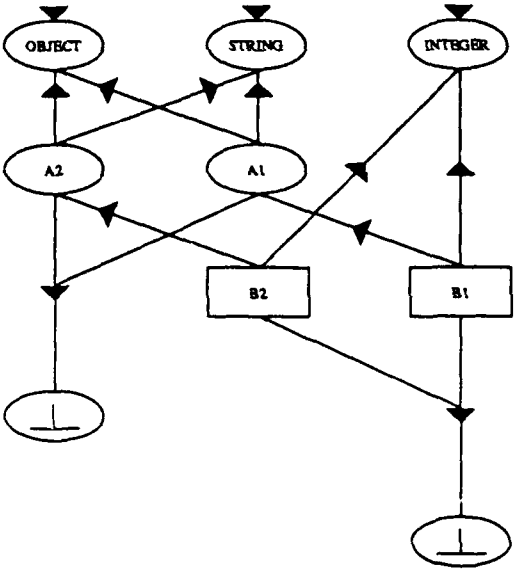
$\langle OBJECT, \{\{\}\}, \{(), (A_1)\}\rangle$
$\langle STRING, \{\{\}\}, \{(), (A_1)\}\rangle$
$\langle INTEGER, \{\{\}\}, \{(), (B_1)\}\rangle$
$\langle A_1, \{\{B_1\}\}, \{(B_1)\}\rangle$
$\langle B_1, \{\{B_1\}\}, \{(B_1)\}\rangle$

Figure 6.13: Newly-defined class B recorded as an *assumption*



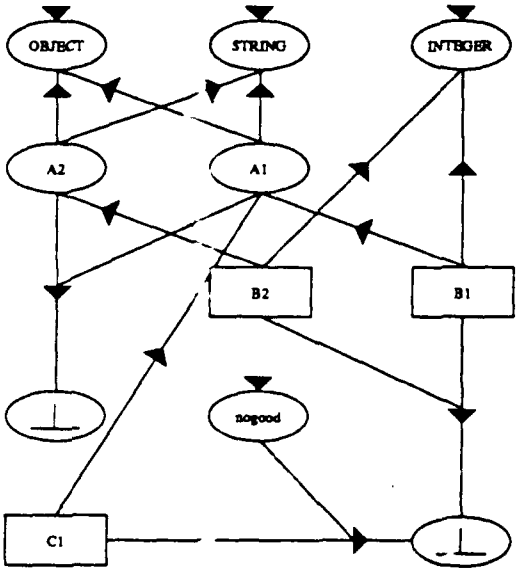
$\langle OBJECT, \{\{\}\}, \{(), (A_1), (A_2)\}\rangle$
$\langle STRING, \{\{\}\}, \{(), (A_1), (A_2)\}\rangle$
$\langle INTEGER, \{\{\}\}, \{(), (B_1)\}\rangle$
$\langle A_1, \{\{B_1\}\}, \{(B_1)\}\rangle$
$\langle B_1, \{\{B_1\}\}, \{(B_1)\}\rangle$
$\langle A_2, \{\{A_2\}\}, \{(A_2)\}\rangle$
$\langle \bot, \{\}, \{(A_1, A_2)\}\rangle$

Figure 6.14: Modified class A recorded as an *assumption*



$\langle OBJECT, \{\{\}\}, \{(), (A_1), (A_2)\}\rangle$
$\langle STRING, \{\{\}\}, \{(), (A_1), (A_2)\}\rangle$
$\langle INTEGER, \{\{\}\}, \{(), (B_1), (B_2)\}\rangle$
$\langle A_1, \{\{B_1\}\}, \{(B_1)\}\rangle$
$\langle B_1, \{\{B_1\}\}, \{(B_1)\}\rangle$
$\langle A_2, \{\{B_2\}\}, \{(B_2)\}\rangle$
$\langle \bot, \{\}, \{(A_1, A_2), (B_1, B_2)\}\rangle$
$\langle B_2, \{\{B_2\}\}, \{(B_2)\}\rangle$

Figure 6.15: Re-compiled class B recorded as an *assumption*



$\langle OBJECT, \{\{\}\}, \{(), (A_1), (A_2)\}\rangle$
$\langle STRING, \{\{\}\}, \{(), (A_1), (A_2)\}\rangle$
$\langle INTEGER, \{\{\}\}, \{(), (B_1), (B_2)\}\rangle$
$\langle A_1, \{\{B_1\}\}, \{(B_1), (C_1)\}\rangle$
$\langle B_1, \{\{B_1\}\}, \{(B_1)\}\rangle$
$\langle A_2, \{\{B_2\}\}, \{(B_2)\}\rangle$
$\langle \bot, \{\}, \{(A_1, A_2), (B_1, B_2), (C_1, nogood)\}\rangle$
$\langle B_2, \{\{B_2\}\}, \{(B_2)\}\rangle$
$\langle C_1, \{\}, \{(C_1)\}\rangle$
$\langle nogood, \{\}, \{()\}\rangle$

Figure 6.16: Inconsistent class C marked as *nogood*

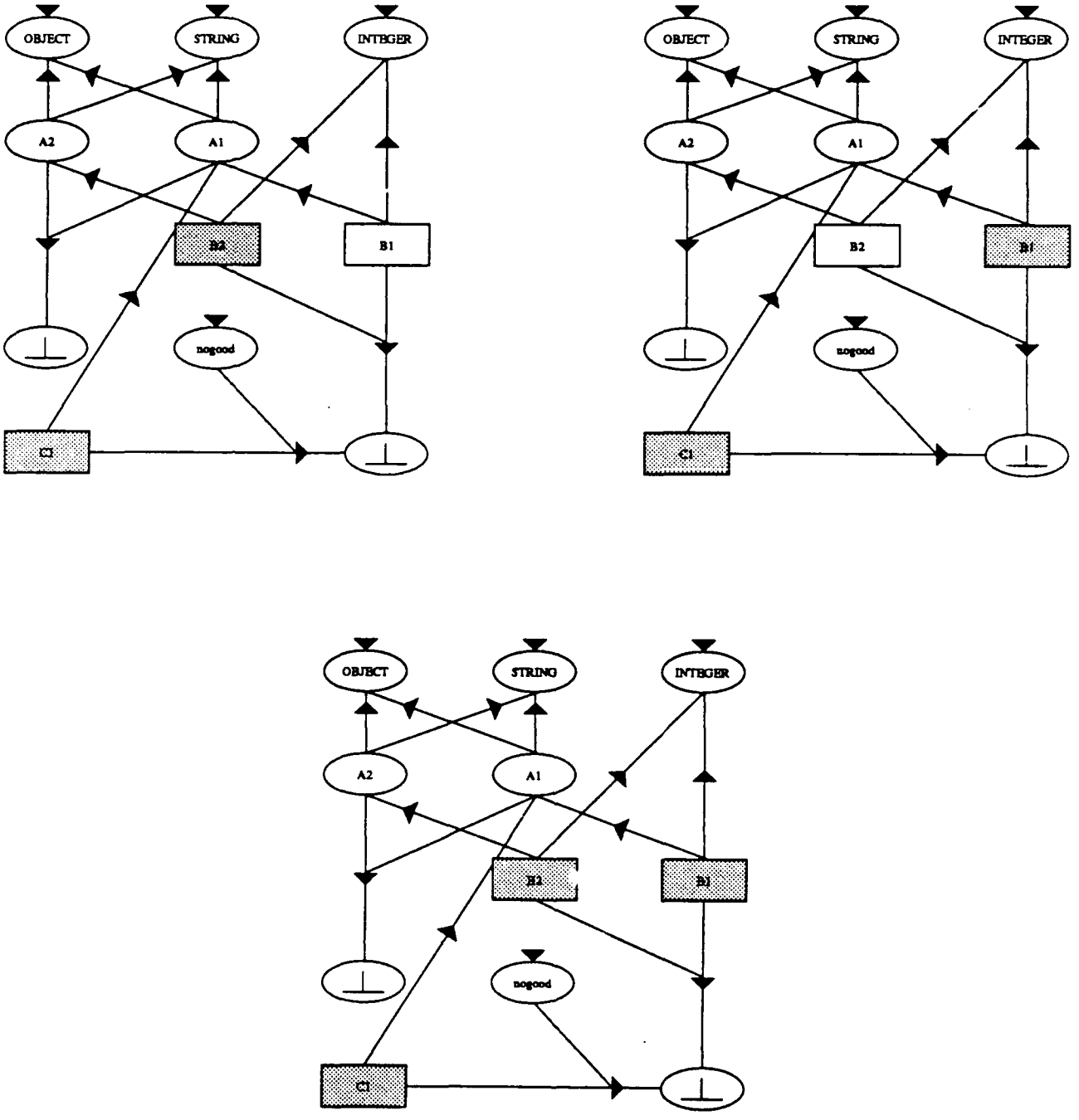


Figure 6.17: Three identifiable schemata

6.5 Further research

In section 6 of this report, we have discussed dictionary support for schema evolution in object-oriented databases. Section 6.1 reviewed major concepts common to many object-oriented data models and established working definitions for terms such as schemata and metadata as they apply to object-oriented databases. Section 6.2 presented two problems associated with schema evolution in object-oriented databases, and section 6.3 reviewed literature on schema evolution for proposed solutions to these problems. Finally, section 6.4 presented our proposal for a data dictionary that is capable of reasoning about evolving schemata and identifying versions of schemata from a collection of versioned classes.

As yet, our proposal is incomplete — further research needs to be conducted in two areas so that such a data dictionary can become a reality. The first area requiring further investigation involves the resolution of ambiguous references during the process of data definition. When the database designer defines a class A which references a class B for which multiple versions $B_1 \ldots B_n$ exist, the data dictionary must know or be told unambiguously which version of class B is being referred to. We are investigating a means by which the labels associated with ATMS data structures can be used to limit the set of potential referents.

The second area for further research involves a method for dealing with instances stored persistently in the database. The automatic conversion of instances proposed by Gemstone researchers in [B+89, PS87] does not make sense in an environment supporting versions of classes which may be used for different applications. The screening approach proposed by Skarra and Zdonik in [SZ86, SZ87] will not always work because it will not always be possible to define semantically meaningful prehandlers and posthandlers. We are investigating Kim and Chou's notion of an access scope for, or set of objects accessible to, a schema version to see if it may by adapted to work when classes are the versioned objects.

7 Active Rule Management in Object-Oriented Databases

7.1 Introduction

Using rules for describing and managing the behavioral aspect of knowledge is one area of research in the fields of deductive databases and an object-oriented paradigm. Interest in this approach has increased significantly in the literature. Integration of an object-oriented paradigm and deductive databases has been influenced by semantic modeling and semantic query processing research [HTY89, HS89, Ngu86, AK89, And87, BS88a, DBM88, Fis87, GK87, KL89a, K+89, KN86, Nie87, SZ86].

Active database research has dealt primarily with integrating production systems with relational databases. In this section, we present an approach to associate rules to object types in an object-oriented model. Since rules are distributed in an object-oriented database, inference is limited either only within a class to which the rules are associated or outside the scope of individual classes by means of *explicit triggers*. Propagation algorithms [HS89, DBM88, Fis87, KL89a] are also executed in a predesignated manner for all users to make use of the same database constraints. In these algorithms, rules are directly activated by explicitly pre-defined triggers.

In this paper, a technique for rule processing by *active inference* will be developed. Our approach extends previous work as follows:

- Every object in an object-oriented data model has a unique object identifier or *oid*. Rules that involve objects can access them by means of the oid. This is in contrast to rule firing in the relational data model where tuples are identified by associative retrieval of attribute values.
- Since a set of rules can be regarded as an object [DBM88], rules can be encapsulated[1]. Then these encapsulated rules may be associated with an object. The rules are associated with a class, rather than being embedded within a class.
- Rule sets are associated directly to classes (or object types). Thus, inference is limited to using those rules that are associated with a class or its inheriting classes. To remove this limitation, a rule schema is constructed which represents the dependencies among the rule sets and associated attributes. This schema provides a *qualitative* explanation of the consequences of invoking one or more rule sets.

The goal of this paper is to propose a scheme for rule specification, management, and processing, by which "*active*"[2] object-oriented databases can be constructed. To do so, rules are represented and depicted in a rule schema in Section 7.2. In Section 7.3, a scheme of rule activation is described. To activate rules, a *meta-message equation* is generated from a rule schema in Section 7.4. Section 7.5 and Section 7.6 discuss rule inference in generalization hierarchies and composite objects, respectively. Finally, we summarize our work and discuss future work.

[1] A rule encapsulated as an object could have methods, attributes, opeations, and invariants which are hidden from other objects.

[2] An *active* rule is a rule that takes one or more actions in response to a database access and update.

7.2 Preliminaries

A rule is a declarative specification about a behavioral aspect [HTY89] expressed as a Horn clause. In this section, the concept of a rule is formulated. Rules are encapsulated as a rule set and the rule set is associated with an object type. Rules are also depicted in a rule schema. Note that a "rule" is used as a generic term in this paper, so we do not distinguish constraints from general production rules. Constraints and production rules are activated in a similar manner. Hence, we consider a general method for rule activation.

7.2.1 Formulae

A key feature of our rule representation is that we are employing *dot-notation*[3] and *object identifiers*. Rules are defined by using the convention of *dot-notation* [Mor86], which specifies functional paths [Shi81] within a schema of object types. The concept of an *oid* is a powerful programming primitive for object-oriented database query processing [AK89]. By associating a message with the *oid* of a receiver, it becomes easier to manage rules in both generalization and aggregation hierarchies in object-oriented databases. We now present several definitions.

Definition 1 (Rule). *We define a rule as a behavioral relationship among attributes of objects. A rule is represented in a first-order logic in which antecedents* $\mathbf{p}$ *imply consequents* $\mathbf{q}$, $\mathbf{p}(e) \Rightarrow \mathbf{q}(e')$, *where* $\mathbf{p}$, $\mathbf{q}$ *are sets of predicates*[4] $\mathbf{p} = \{p_1, p_2, ..., p_n\}$ *and* $\mathbf{q} = \{q_1, q_2, ..., q_m\}$, *respectively, and* e, *and* e' *are sets of arguments.*

The arguments of predicates are called *function terms* and take the form

class.attribute(oid)

where an *oid* is either a constant or a variable.

For example, Engineer.degree(Smith) is a function term. Function terms play the role of a variable, denoted by a small letter e, and so they can be bound to particular values. The values can be obtained by message sending, to be defined below.

Definition 2 (Function term as a message). *We denote by class.attribute*(oid), *abreviated by* $E(x)$[5], *a message requesting a data retrieval operation which is received by an object with the object identifier oid, where a constant* E *denotes a functional path* $(e.g., class.attribute)$ *and the variable* x *denotes an oid.*

Consider a company database depicted in Table 7.1. The function term Engineer.degree(Smith) as a message returns a value "MS."

[3] Every pair of objects can be combined through a pairwise-association operator (denoted by ·) into another object.
[4] A predicate denotes true or false, that is, $p(A)$ is either true or false.
[5] A function term is a variable e which can be bound by an attribute of an oid x, i.e., $E(x)$.

Table 7.1: Company Database - Employee and Engineer Object Types

Employee

oid	name	dependents	salary	taxRate
Smith	Smith Turner	2	55000	10%
James	James Jeffrey	0	60000	10%
Tom	Tom Parker	1	32000	8%

Engineer

oid	degree	yearsOfExperience	worksOn
Smith	MS	9	DARPA1
James	PhD	10	DARPA1
Tom	BA	3	RADC3 DARPA1

Definition 3 (Substitution). *A substitution is any finite set of associations between variables and assertions. A variable w can be bound by substituting by its binding.*

For example, suppose there is a rule associated with the Employee type of Table 7.1 and Figure 7.1 such that: an engineer's salary should be over forty thousand dollars if he has obtained a PhD degree.

$$\text{Engineer.degree}(x) = \text{"PhD"} \Rightarrow \text{Engineer.salary}(x) > \$\ 40{,}000$$

The first predicate "=" in this rule contains a function term "Engineer.degree(x)." The function term itself is a message being sent to retrieve data from an object X by Definition 2. By this message sending, the value can be substituted for the attribute "degree" of X by Definition 3. Therefore, Engineer.degree(Smith) is evaluated to "MS", where Smith is an *oid* for the employee named Smith from Table 7.1. This value, then, is tested with the value "PhD" by the predicate "=." Similarly, the predicate in the consequent part can be evaluated. For the case of Smith, the rule does not apply. However, for James, who does have a PhD, his salary should be greater than \$40,000 and indeed it is \$60,000. Note that the rule can be viewed as an integrity constraint. The above description indicates the rule matching and execution process.

Definition 4 (Rule set). *Rules can be grouped into a "rule set".*

For example, rules concerning "salary" can be grouped into the rule set "salary-rules."

RULESET **salary-rules** (1)
Engineer.degree(x) = "PhD" $\Rightarrow$ Engineer.salary(x) > \$ 40,000;
Engineer.degree(x) = "MS" $\Rightarrow$ Engineer.salary(x) > \$ 35,000;
END salary-rules.

There are several advantages to using rule sets. They are:

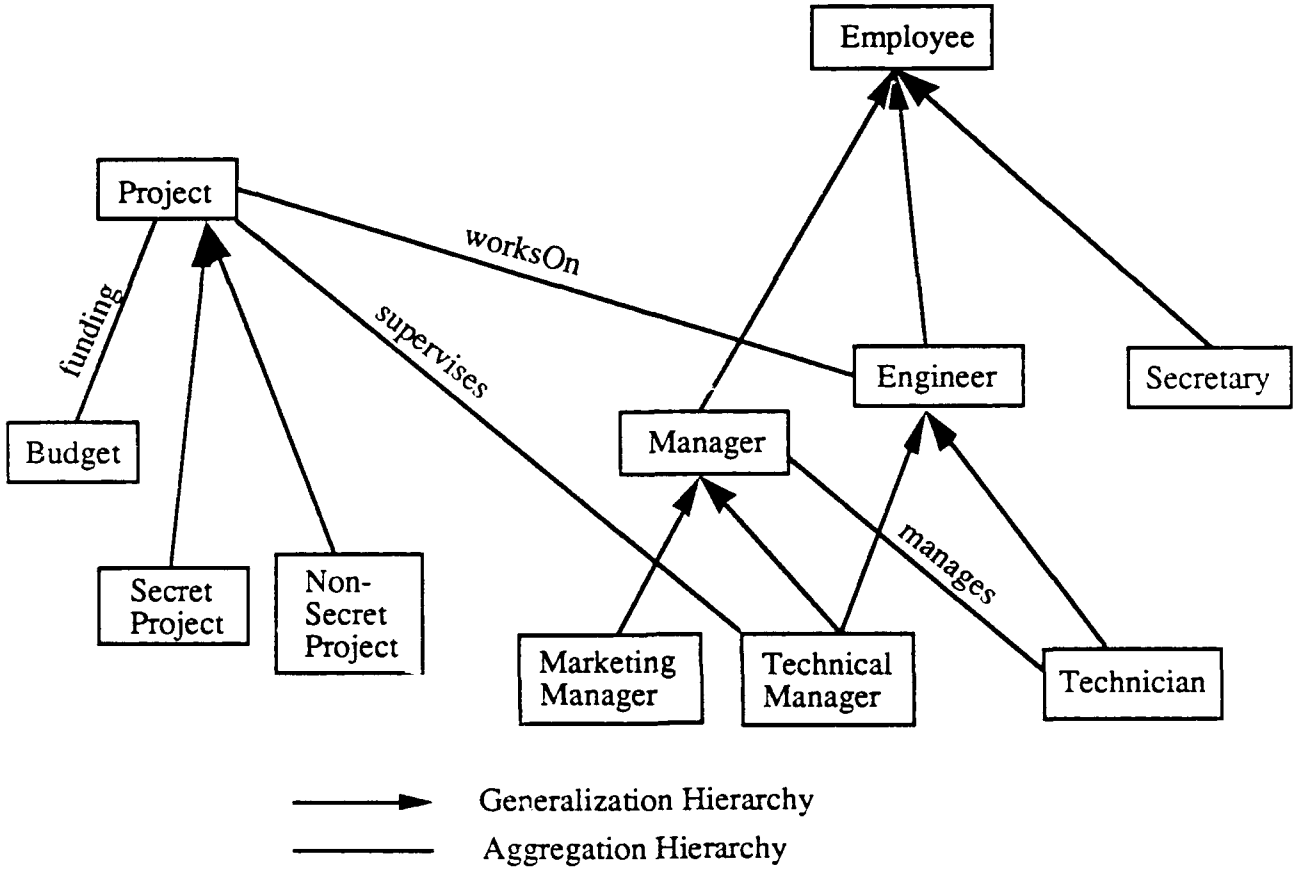


Figure 7.1: A Schema of a Company Database

- Rules can be directly related and associated with a particular object or an attribute of the object.
- The scope of rule searching can be restricted to a single rule set associated with an object.
- Rules can be managed efficiently since they are grouped and are associated with an object type. Thus, the rule sets will be inherited and thereby shared by all "children" of that object type as opposed to the rule sets being replicated in those objects.

7.2.2 Rule Taxonomy

In general, procedural components can be associated with either a method or as an active value in the object-oriented paradigm. Likewise, rules associated with a class can be classified either as *Rules as a method* or *Rules as an active value.*

Rules in the rule set **salary-rules** can be associated with a method. This is an example of *rules as a method*:

```
OBJECTTYPE      Employee                              (2)
ATTRIBUTE       name          VALUE;
                dependents    VALUE;
                salary        VALUE;
                taxRate       VALUE;
METHOD          salary-rules;
END Employee.
```

Rules as an active value are associated with an attribute of an object type. For example, "**salary-rules**" can be declared in an attribute slot:

```
OBJECTTYPE      Employee                              (3)
ATTRIBUTE       name          VALUE;
                dependents    VALUE;
                salary        salary-rules;
                taxRate       VALUE;
END Employee.
```

The distinguishing characteristics between these two types will be discussed in greater detail in later sections.

7.2.3 Rule Schema

A rule schema is a graph which expresses dependencies among a collection of attributes of objects. It consists of nodes connected by edges. A node represents an attribute E, while an edge represents

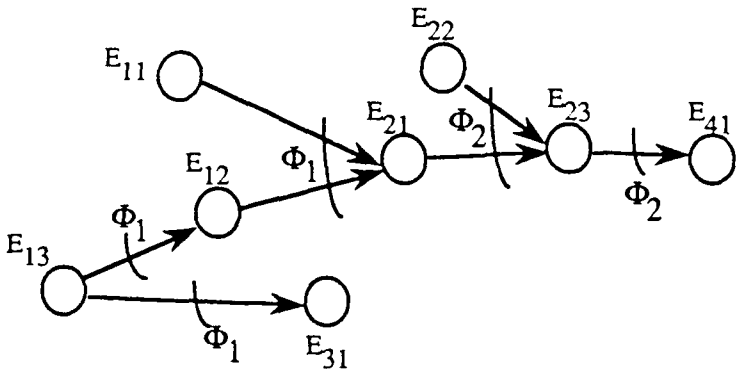


Figure 7.2: Graphic Depiction of a Rule Schema - Small Example

a relationship among attributes. Note that a rule is described in terms of function terms e (i.e., $E(x)$), but a rule schema is constructed in terms of attributes E.

Definition 5 (Rule Schema). *The rule schema is defined by a 3-tuple* $\mathbf{M} = (\mathbf{E}, \mathbf{W}, \Phi)$, *where* $\mathbf{E}$ *is a set of nodes (i.e., attributes),* $\mathbf{W}$ *is a set of real world values.* Φ *is a set of edges (rulesets) where* $\Phi(\mathbf{E}, \mathbf{W}) = \mathbf{E}'$ *is a set of state transition functions. The output of* $\Phi(\mathbf{E}, \mathbf{W})$ *is a set of attributes* $\mathbf{E}'$.

That is, $\Phi = (\Phi_1, \Phi_2, ..., \Phi_n)$ with corresponding substituted values $\mathbf{W} = (W_1, W_2, ..., W_l)$ transits from nodes $\mathbf{E} = (E_1, E_2, ..., E_m)$ to another set of nodes $\mathbf{E}'$. Consider, for example, the rule schema $\mathbf{M}$ depicted in Figure 7.2. There are two rule sets, $\Phi_1(E_1, W_1)$ and $\Phi_2(E_2, W_2)$. The set of nodes (attributes) $\mathbf{E}$ consists of $E_1 = (E_{11}, E_{12}, E_{13})$ and $E_2 = (E_{21}, E_{22}, E_{23})$. The set of real world values $\mathbf{W}$ consists of $W_1 = (W_{11}, W_{12}, W_{13})$ and $W_2 = (W_{21}, W_{22}, W_{23})$. The output of this set of rule sets will be $\Phi_1(E_1, W_1) = E_1'$ and $\Phi_2(E_2, W_2) = E_2'$, where $E_1' = (E_{21}, E_{31})$ and $E_2' = (E_{41})$.

7.3 Rule Activation

There are three issues in rule processing: activation of a rule set; match or selection of a rule from the rule set; and execution of the matched or selected rule. Since rules are grouped into a rule set and rule sets are associated with classes, the first step in rule processing is to activate the rule set of that class. Then, rules in that activated rule set are matched and selected with a particular rule eventually being fired. This section deals with activation of a rule set. Schemes for activating each rule type are described. Note that both types are initially activated by sending messages.

7.3.1 Activation of rules as a method

In a manner similar to procedure calls, a message is sent to an object. When the message is matched with a method of the object, the method is activated. Such a message sending is the standard way

to invoke a method in an object-oriented paradigm. Its syntax is

$$\Phi(oid, argument)$$

where Φ is a name of a rule set as a method which an oid invokes. The argument is optional.

In object-oriented databases, this message can be generated from a database query. To invoke a corresponding rule set for a query, let us consider the following definition.

Definition 6 (Predicate Submodel). *Let p_i be a predicate and M_i be a model of p_i, a set of objects satisfying p_i. A model M_1 of p_1 is a predicate submodel of a model M_2 of p_2 iff*

1. *p_1 and p_2 are identical by appropriate substitutions for their variables, and*
2. *M_1 is a subset of, or is equal to M_2.*

Theorem 1 (Rule set activation to an oid in a query). *If a query is a predicate submodel of a rule set, then the rule set can be activated.*

Proof. Suppose the rule set is not activated. Then, the objects satisfying the rule set must be exclusively disjoint with the objects satisfying a query. But, this contradicts our premise that the query is a predicate submodel of the rule set. □

By Definition 6, a rule set which contains a rule $p(x) \Rightarrow q(y)$ is activated when a query $p(A) \Rightarrow$ is issued. This occurs because $p(x)$, by substituting A for x (denoting $p(x)\{x/A\}$), is identical to $p(A)$. For example, suppose the following query is issued:

Employee.degree(Smith) is updated to PhD $\Rightarrow$

Then, a rule in a rule set **salary-rules**, (2) in Section 7.2.2, is identical to the query by the substitution: Employee.degree(x) {x / Smith} $\equiv$ Employee.degree(Smith). A set of models for predicate Employee.degree(Smith) is a subset of that for Employee.degree(x). From these two conditions, the query is a predicate submodel of a predicate Employee.degree(x). By taking an oid $Smith$ from the query, then, a message is sent to the object "Smith": salary-rules(Smith). This is a very simple case. In many cases, however, oids are not taken from queries, rather they need to be obtained by the following process.

Definition 7 (Object Identification). *Object identification is a process which identifies a set of $(attribute, value)$ pairs as a corresponding $oid(s)$. That is, (E, W) corresponds to an oid or oids. A set of oids identified to a predicate $p(E(x)\ C)$ is $\{x \mid \exists x\ p(E(x)\ C)\}$.*

By the *object identification*, each tuple is represented as a set of (attribute, value) pairs. For example, a tuple of Smith is represented as (degree, "MS"), (dependents 2), and so on as in Table 7.1.

Let's take another example. Suppose a query asks for any employee who has earned a PhD degree:

Employee.degree(x) = PhD $\Rightarrow$

Selection of rule sets is the same as above. Then, we need to obtain oids to which a message is sent. The oids are obtained as the definition above: $\{x \mid \exists x$ Employee.degree(x) = PhD $\} \equiv \{James\}$. Therefore, a message salary-rules($James$) is sent.

Consequently, rule set activation requires a notion as to *what message* is sent to *which object* and in *what order*. Keeping this in mind, an inference mechanism in object-oriented databases is proposed as following steps:

Rule Inference Process

Step 1 A rule set, of which a query is a predicate submodel, is selected.

Step 2 The appropriate oid(s) is found. It is either identical to an oid in a query or obtained from the process of object identification. This oid is not changed unless composite objects exist along a function path.

Step 3 Messages, in the form of $ruleset(oid)$, are sent to the object with the oid from the above step in the order specified by a *meta-message equation.*

Step 1 and Step 2 have already been explained in this section. Step 3 will be discussed in Section 7.4.

7.3.2 Activation of rules as an active value

When a rule-set is associated with an attribute as an active value, then only that rule-set is activated when the attribute of the object containing that attribute is accessed. For example, "**salary-rules**" in (3) above is activated when the salary of an object is retrieved or updated as in the queries $Employee.salary(Smith)$ and $Employee.salary(Smith)$ *is* \$56,000, respectively. The syntax of *rules as an active value*, as defined by Definition 7, is simply

$E(x)$

The main difference between activation of *rules as an active value* and *rules as a method* is that all rule sets in an object which are applicable to a query are activated in *rules as a method*, while only those rule sets associated with a specific attribute are activated in *rules as an active value*. Activation in response to data access (i.e., *rules as an active value*) is, thus, quite simple. Therefore, later sections will emphasize activation of *rules as a method* rather than *rules as an active value*.

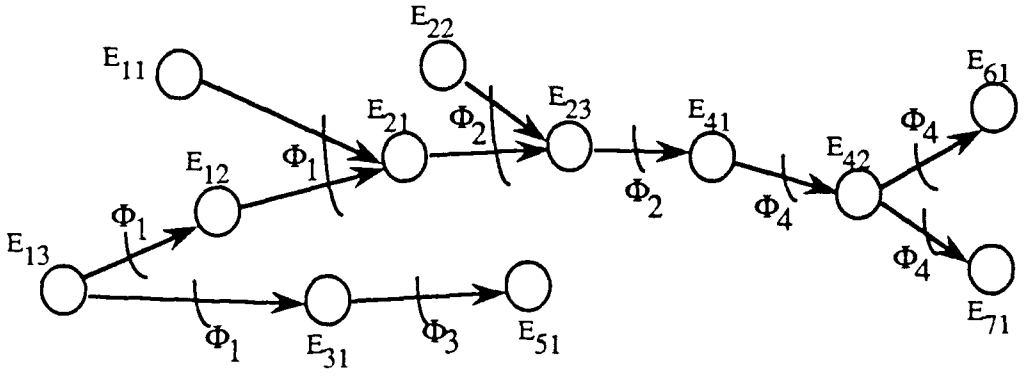


Figure 7.3: Graphic Depiction of a Rule Schema - Extended Example

7.4 Meta-Message Equation

Rule processing within a class may not be sufficient to provide a solution to a query. Reasoning in object-oriented databases frequently requires rule sets which are beyond the scope of a single object type. Therefore, a number of rule sets may be needed to answer a query. For an inference that spans more than one object type, a meta-message equation is generated. In this section, we investigate a mechanism for active message sending. Let's consider the following rule sets:

Rule set 1 $\Phi_1 : p_1(e_1) \Rightarrow q_1(e_1')$
Rule set 2 $\Phi_2 : p_2(e_2) \Rightarrow q_2(e_2')$
Rule set 3 $\Phi_3 : p_3(e_3) \Rightarrow q_3(e_3')$
Rule set 4 $\Phi_4 : p_4(e_4) \Rightarrow q_4(e_4')$

where the arguments e_i and e_i' are function terms $E_i(x)$ and $E_i'(x)$ respectively. Figure 7.3 is a graphic depiction of an example rule schema which consists of the equations:

$$\Phi_1(E_1, W_1) = E_1'; \quad \Phi_2(E_2, W_2) = E_2';$$
$$\Phi_3(E_3, W_3) = E_3'; \quad \Phi_4(E_4, W_4) = E_4',$$

where attributes $E_1 = (E_{11}, E_{12}, E_{13})$, $E_2 = (E_{21}, E_{22}, E_{23})$, $E_3 = (E_{31})$, $E_4 = (E_{41}, E_{42})$, $E_1' = (E_{21}, E_{31})$, $E_2' = (E_{41})$, $E_3' = (E_{51})$, $E_4' = (E_{61}, E_{71})$, attribute values $W_1 = (W_{11}, W_{12}, W_{13})$, $W_2 = (W_{21}, W_{22}, W_{23})$, $W_3 = (W_{31})$, $W_4 = (W_{41}, W_{42})$.

7.4.1 Equation Generation

Suppose a query $p_1(E_{13}(oid_1)\ C) \Rightarrow$ is issued. The query concerns attribute $E_{13}(oid_1)$ as well as p_1. Thus, the query is a predicate submodel of $p_1(E_{13}(x)\ C)$. In other words, $p_1(E_{13}(x)\ C)\{x/oid_1\} \equiv p_1(E_{13}(oid_1)\ C)$. Then, by Theorem 1, rule set Φ_1 can be activated after making the substitution $\{x/oid_1\}$. Now, in order to determine the effect of activating Φ_1, rule sets in Figure 7.3 are collected and the equation grows in a bottom-up manner by walking along the edges until no more edges are available. In our example, this process starts from the innermost rule set Φ_1 and ends with Φ_3 and Φ_4, as can be seen from Figure 7.3. The relevant attribute values in the above rule schema example are W_2, W_3 and W_4. The equations that result from this process are:

$$\Phi_4(\Phi_2(\Phi_1(E_1, W_1), W_2), W_4) = E_{61}$$
$$\Phi_4(\Phi_2(\Phi_1(E_1, W_1), W_2), W_4) = E_{71}$$
$$\Phi_3(\Phi_1(E_1, W_1), W_3) = E_{51}$$

We call these equations *meta-message equations*. A meta-message equation directs messages to be sent to objects in the order in which rule sets are activated. This equation, thus, enables qualitative reasoning which, according to the query, makes it possible to follow an inference chain to attribute E_{61}, E_{71}, or E_{51}.

7.4.2 Message Generation

A meta-message equation denotes an order for sending messages among objects. Rule sets are activated in a bottom-up manner, starting with the innermost message; in the same manner that the equations were generated.

Let's consider our first meta-message equation $\Phi_4(\Phi_2(\Phi_1(E_1, W_1), W_2), W_4) = E_{61}$. The query specified an oid oid_1, so the innermost rule schema, (E_{13}, W_{13}) is identical to oid_1 by Definition 7. That is, $\Phi_1(E_1, W_1) = \Phi_1(oid_1)$ where the generated message is:

Message 1: $\Phi_1(oid_1)$

Now, by Definition 5 and from our example rule schema, $\Phi_1(oid_1) = E_{21}$. When this substitution is made, the equation becomes $\Phi_4(\Phi_2(E_{21}, W_{21}), W_4) = E_{61}$, where W_{21} are the real world values of W_2 that are specific to E_{21}. The next message is generated from the new innermost rule schema, $\Phi_2(E_{21}, W_{21})$, and the appropriate oid. This message is:

Message 2: $\Phi_2(oid_2)$

Again, by Definition 5 and from our example rule schema, $\Phi_2(oid_2) = E_{41}$. When this substitution is made, the equation becomes $\Phi_4(E_{41}, W_{41}) = E_{61}$ where W_{41} are the real world values of W_4 that are specific to E_{41}.

Message 3: $\Phi_4(oid_4)$

Notice that these same messages would be generated from the second equation. Using the third equation, the resulting messages [6] are:

Message 1: $\Phi_1(oid_1)$
Message 2: $\Phi_3(oid_3)$

This rule activation mechanism is discussed in the context of both generalization and aggregation inheritances in the following sections.

7.5 Rule Inference in Generalization Hierarchies

In this section, activation of a rule set will be discussed for generalization hierarchies. For example, there are two rule sets [7] which are depicted in Figure 7.4.

RULESET **salary-rule2**
Engineer.yearsOfExperience(x) $\geq$ 10 $\Rightarrow$ Engineer.salary(x) > \$ 50,000;
END salary-rule2.

RULESET **tax-rules**
Employee.salary(x) > \$ 50,000 $\wedge$ Employee.dependents(x) = 2
$\Rightarrow$ Employee.taxRate(x) = 10%;
Employee.salary(x) < \$ 30,000 $\wedge$ Employee.dependents(x) = 2
$\Rightarrow$ Employee.taxRate(x) = 2%;
END tax-rules.

For example, suppose a query is issued to update technical engineer (i.e., *technician*) Smith's years of experience to 10 years. Recall from Table 7.1 that he currently has 9 years of experience and earns a salary of \$55,000: yearsOfExperience(Smith) is updated to 10 $\Rightarrow$.

From the rule schema in Figure 7.4, the following meta- equation is generated.

tax-rules (salary-rule2 (yearsOfExperience, 10), \$55.000) = taxRate

[6] Notice that the order in which messages are generated is determined by the selected equation and not by the order in which equations are selected. The order in which the equations are selected is relevant, but is not addressed in this report.

[7] Note that salary information is associated with the class Employee while yearsOfExperience is associated with Engineer. The function term "Engineer.Salary(x) *is_a* Employee.salary(x)" is implicit due to the generalization hierarchies.

Table 7.2: Company Database - Extended Example

Budget

oid	amount
DARPA1	250000
RADC3	30000
RADC4	300000

Project

oid	contractor	type
DARPA1	Henry Co	B
RADC3	Smart Co	D
RADC4	Beaty Co	A

SecretProject

oid	class
DARPA1	top secret
RADC3	confidential
RADC4	top secret

TechnicalManager

oid	supervises	clearanceLevel
James	DARPA1	1
Tom	RADC4 DARPA1	1

Technician

oid	managed_by
Smith	James

where the attribute values are taken from Table 7.1. Since a query is issued only for an object "Smith," we can use the oid of Smith.

tax-rules (salary-rule2 (Smith), $55,000) = taxRate

Because a new object identification is not necessary, we still keep the same oid as above:

tax-rules (Smith) = taxRate

Thus, two rules, salary-rule2 and tax-rules, are activated sequentially for Smith by the following messages.

Message 1: salary-rule2 (Smith)
Message 2: tax-rules (Smith)

By sending messages and activating rules, we deduce that the tax rate for technician Smith is 10%.

7.6 Rule Inference in Composite Objects

In object-oriented databases, an object may reference a number of other objects. This relationship is called the "IS-PART-OF" relationship [KBG89] and allows composite objects to be represented as aggregations of other objects. The composite object can be represented in an aggregation hierarchy. In this section, two types of composite objects, *dependent* and *independent*, are considered.

7.6.1 Inference in Dependent Composite Objects

We say that O is a *dependent* composite object to O' if the existence of O depends on the existence of O'. Rule processing in dependent composite objects is simple, so that the technique used in generalization hierarchies can be applied. A difference is that a function term is represented by a

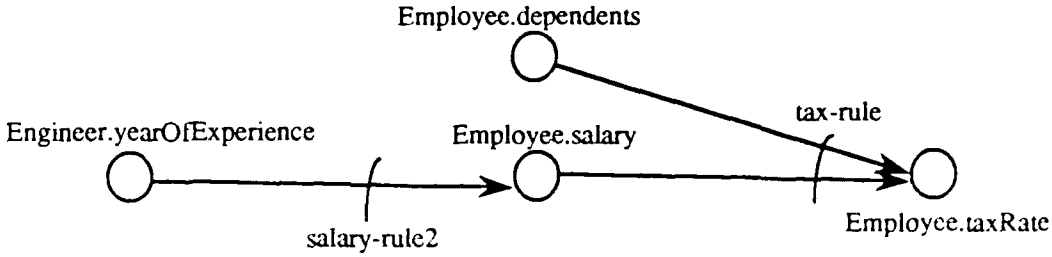


Figure 7.4: A Rule Schema of Object *Technician*

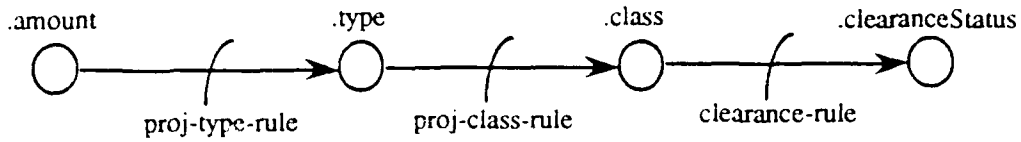


Figure 7.5: A Rule Schema of Object *Secret Project*

functional path, similar to [Mor86, Shi81].

RULESET **proj-type-rules**
Project.funding.Budget.amount(x) > $ 200,000 ⇒ Project.type(x) = "B";
Project.funding.Budget.amount(x) < $ 50,000 ⇒ Project.type(x) = "D";
END proj-type-rules.

RULESET **proj-class-rule**
SecretProject.type(x) ≥ "B" ⇒ SecretProject.class(x) = "top secret";
END proj-class-rule.

For example, suppose a query asks the security class of all secret projects whose budget is over $200,000: Budget.amount(x) > 200,000 ⇒ SecretProject.class(x). By applying the definition of predicate submodel and thus Budget.amount(x) ≡ Budget.amount(x), the following meta-message equation is obtained from the rule schema in Figure 7.5:

proj-class-rule (proj-type-rules (Project.funding.Budget.amount, w_1), w_2) = SecretProject.class

Since oids are not provided by the query, oids can be obtained by the definition of object identification. By Definition 7 they are $\{x \mid \exists x$ Budget.amount$(x) >$ \$200,000 $\} \equiv \{DARPA1, RADC4\}$. When we apply the above meta-message equation to project DARPA1 from the obtained oid set, we get the equation:

proj-class-rule (proj-type-rules (DARPA1), B) = SecretProject.class

From this instantiated equation, rule sets are activated by sending messages for the project instance of DARPA1:

Message 1: proj-type-rules (DARPA1)
Message 2: proj-class-rule (DARPA1)

Finally, we conclude that DARPA1 is a top secret project. Similarly, we have another meta-message equation by applying to project RADC4:

proj-class-rule (proj-type-rules (RADC4), A) = SecretProject.class

By sending messages to RADC4, (1) proj-type-rules(RADC4) and (2) proj-class-rule(RADC4), it is concluded that RADC4 is a top secret project.

7.6.2 Inference in Independent Composite Objects

Let object types O and O' be composite object types. O is said to be *independent* of O', if the existence of O does not depend on the existence of O'.

First, dot notation is useful in describing the rule sets of independent composite objects, playing the role of the "join" operation in relational databases. A functional path along independent composite objects needs more than one function term. From the following example, we see that the class of secret projects which are supervised by a technical manager can be expressed as:

$$\text{TechnicalManager.supervises}(x).\text{SecretProject.class}(y) = C$$

Because the existence of independent composite objects are by definition independent on the existence of other composite objects, oids must be changed along the functional path of aggregation hierarchies. In order to identify the oid of independent composite objects, we may use the following definition.

Lemma 1 (Object Identification in Composite Objects). *Let a function path be* $E_1(x).E_2(y).E_3(z) = C$. *Then, oids* X *of the independent composite objects can be obtained using the following equations.*

$$Z = \{z \mid \exists z\ E_3(z) = C\}$$
$$Y = \{E_2(y) \mid \forall y\ y \in Z\}$$
$$X = \{E_3(x) \mid \forall x\ x \in Y\}$$

From our TechnicalManager example above, the set of oids Y is $Y = \{y \mid \exists$ SecretProject.class(y) $= C\}$. Once the set Y is obtained, the set of oids of composite objects, X, can be found: $\{$ TechnicalManager.supervises$(x) \mid \forall x\ x \in Y\}$.

With this notion in mind, let us consider an example of rule set activation. Suppose that in addition to the rule sets of the previous section we have the rule set:

```
RULESET                    clearance-rules
TechnicalManager.supervises(x).SecretProject.class(y) = "top secret"
⇒ TechnicalManager.clearanceLevel(x) = "1st";
TechnicalManager.supervises(x).SecretProject.class(y) = "confidential"
⇒ TechnicalManager.clearanceLevel(x) = "3rd";
END clearance-rules.
```

Consider rule schema in Figure 7.5. Now suppose we want to obtain the clearance of the manager of a secret-project DARPA1. The following meta-message equation can be generated:

clearance-rules (proj-class-rule (proj-type-rules (Project.funding.Budget.amount, 200,000), A), w) = clearanceStatus

From this equation, rule sets are activated by sending messages to the object which has the oid of DARPA1 (which is an oid taken from the query). The first two messages generated from the two innermost meta-message equations are:

Message 1: proj-type-rules (DARPA1)
Message 2: proj-class-rule (DARPA1)

The rest of the equation is:

clearance-rules (TechnicalManager.supervises, w) = clearanceStatus

Given that $Y = \{DARPA1\}$ from the query, by using Lemma 1, the desired oids are:

$$\{ \text{TechnicalManager.supervises}(x) \mid \forall x \; x \in Y \} = \{James, Tom\}.$$

Therefore, a message can be sent to both objects of oids "James, Tom":

Message 3: clearance-rules (James)
Message 4: clearance-rules (Tom)

We conclude that a technical manager James' clearance is 1 and Tom's clearance is 3.

7.7 Summary

A technique for knowledge processing in object-oriented databases is developed in this report. Rules use a *dot notation* and an *oid* of an object. This representation is suitable to the features of an object oriented paradigm including generalization and composite objects. Rules are grouped into rule sets and the rule sets are depicted in a rule schema. A rule set is regarded as a method or an active value in object-oriented databases, and so two notions of rule activation are described. *Meta-message equations* can be generated from a rule schema. Based on each meta-message equation, rules are automatically and sequentially activated.

8 Hypermedia Requirements for Active Dictionaries

8.1 Introduction

In a multimedia application one typically interacts with the data through a workstation with a multi-windowing capability. The user will likely be interested in displaying and interacting with several multimedia types. For instance, in a medical application concerning the heart, the user may have a window open that displays one or more views of the heart, another window that shows the location of the heart within the body, a window with textual data, another window that shows the position of the user within the data structure and another window with motion video on some other aspect such as a beating heart. In addition, voice data can be presented to the user at appropriate times.

In order to orchestrate the presentation of the information, one must have a methodology for drawing the proper data from storage, integrating it, synchronizing the presentation to the user, and finally displaying it. This takes a multilevel object-oriented system. For instance, the actual data can be stored as sub-objects with higher level entities representing various combinations of integrated and synchronized objects. The most straight-forward approach is to pre-synchronize the data, such as the sound and picture in a movie. While such data can be pre-synchronized the general problem requires the tools and techniques to develop the presentation in real time.

We have considered how the IRDS might be extended to handle conventional database data, knowledge, text, imagery, spatial data, temporal data, engineering data and CAD/CAM data. We use the concept of a multimedia document to illustrate the ideas. For instance, as a brief introduction, consider imagery and video. Traditionally, one extracts information from an image and places it along with text in a structured record. The actual image, in digital form, is stored in a file on a disk and retrieved as needed. In this case there is little difference between imagery and other types of data. However, if we extract some special information about the image this may add to the requirements of the data dictionary. For instance, there may be some particular constraints about the intensity levels within the image. This adds active dictionary entries at the data level. Also, there may be constraints over classes of images which yield entries at the meta or meta-meta level. Also, they may be knowledge entries regarding the imagery that go beyond the capabilities of the current IRDS structure.

With regard to video, new dimensions are added. At a basic level video is just the presentation of many images in rapid sequence. However, the important aspects are contained in the relationships of various images that are not necessarily successive, that is, a particular object appears in a range of frames. This causes the interrelationship between any two objects to become much more complex.

As with imagery, one forms a structured record with text to describe the video. So, from the point of view of an active dictionary, video is similar to text and data in this regard but more complex when higher levels of an active dictionary are concerned. We now elaborate on these ideas.

8.2 Multimedia document

In order to assess the impact of multimedia data on future versions of the IRDS, we have chosen an example that involves a general structure for a multimedia document. The structure is shown in Figure 8.1 and follows no particular object-oriented conventions, but is similar to the diagrammatic

conventions of the Knowledge Data Model described in section 4.2. In particular, the diagram uses solid lines to indicate the "is-part-of" relationship of aggregate, or composite object types. Relationship types associated with the Entity-Relationship model are depicted as diamonds.

The document can be divided into two general parts, the body and illustrations. The body refers mainly to textual data while the illustration part involves all other types. The body includes header and section information that are further divided into such other information as logo and edition, subsections and sections, titles and paragraphs.

The general structure for this part of the multimedia document will most likely exist at the schema and database levels of the IRDS. The actual data that comprise the paragraphs would be stored in the database. The header information may involve other data types such as images (logo) which will be discussed later. Another aspect of the body of the document concerns other data. These data may include a variety of additional types at a higher level of abstraction. For instance, the document may have had several revisions which leads to temporal data regarding the versions. Also, domain specific data may be added as well as general constructs. Most of these data apply to a level higher than the current IRDS levels. However, the constructs of the IRDS's Basic Functional Schema can be used to define new entity types that could be used to represent this high-level meta-data.

Another aspect of the multimedia document structure are relationships. Shown in Figure 8.1 between body and illustration is the relationship type BI. This indicates that there may be some connection between these two aspects of a document. For instance, an image or drawing may have a connection with a particular fragment of text. This type of information may be described at the schema and database level of the IRDS. Also, there are a number of other relationships among data at lower levels of the diagram but they are not shown since this would make the diagram too complex. Under illustration three additional multimedia types are given, these are imagery, video and voice. Each of these types will be discussed in turn.

8.2.1 Image data

The first type we will consider is image data. This category includes a variety of types including engineering drawings, charts, maps, photographs, x-rays, and the like. (While most people would agree that the term image is more restrictive than used here, there is no general agreement upon a term that describes this category so we will use image.) There are three major ways of storing this type of data, scan line analog form, a bit map representation and a vector representation. In the scan line representation, data are stored as modulations of analog signals and are generally treated as a whole image. That is, one can reproduce the entire image at will but cannot manipulate individual parts of the image. In the bit map representation, individual pixels are represented in digital form. For instance, a pixel with color information can be represented in 24 bits, 8 bits each for red, green and blue. These data can now be manipulated individually or in groups thus allowing further processing to be performed. However, the amount of digital data for one image can be quite large; a megabyte for a digitized standard TV frame or as high as several gigabytes for a high quality medical image. The third type of data in this category is vector data. In this case, one has a set of standard representations and locates many of these together at various locations to form the required image. These types of images are common in the case of computer generated images.

At one level all of the above categories contribute files of data that must be managed as a group. That is, one often wants to view the entire image without further manipulation. Thus, the structure of these data can be managed at the schema level of the IRDS and the actual data stored as a file. This is shown as the Image-Body in Figure 8.1. Considering a higher level of the IRDS one needs to create and manage a structured record about the image. Under current implementations this record type will be all text and can be managed accordingly. This is shown as the Image Heater in Figure 8.1. However, in future systems, sample data from the images, such as swatches, would become part of the structured record. There are many reasons to include such data one of which is that the user may want to take a "quick look" at the data in order to help make the decision as to whether further data are needed. This inclusion of image swatches at the directory level will add an additional element of complexity to the IRDS.

At a higher level of complexity one often wants to extract features from an image. Depending upon the particular image involved, the features could be managed at the meta schema or schema level of the IRDS. For instance, if one wants to know which images have trees in them then it would not be difficult to include suitable fields in a structured record to indicate the presence of trees. However, if more complex information were required, such as the shape and size of particular objects in relationship to other objects, then this would be more difficult if not impossible to manage with the current IRDS. In addition to the type of data discussed above, there are operations that are performed on the image that may not pertain to particular features. For instance, one may be interested in color enhancements of the image, compressing the image for use in index generation and processing or, in general, processing the image in some way and keeping various versions of the image. The types of operations described in this paragraph are representative of the types of operations indicated in the third block under image in Figure 8.1 that is, other image data, information and knowledge.

It is clear that the IRDS can handle certain forms of image data. However, its capabilities are exceeded rather quickly when knowledge and certain types of meta data, as described above, are considered. This is not surprising since the IRDS wasn't really designed for such data. However, it is also clear that considerably more study needs to be undertaken to develop an IRDS structure that will adequately deal with image data.

8.2.2 Audio data

Audio data can appear in a variety of ways in a multimedia environment. Various pieces of music or speech can be digitized and stored in the multimedia database. Voice annotations can be collected in conjunction with the analysis of image or video data. For example, a radiologist can input voice annotations about various sections of a medical image during analysis thus adding data to a permanent record.

In discussing audio, we will use the example of the Belfer Audio Laboratory and Archive at Syracuse University which possesses a collection of 1.2 million sound recordings of various types as well as images of each recording. The laboratory possesses high quality digitized audio concerning much of the collection. In addition, lower quality sound is also stored to provide initial screening for the user.

A structured database record exists for each of the recordings as well as the image of the sound recording. The image provides the user with information that cannot readily be stored in the

structured record. Such information includes pictures, drawings, etc that appear on the recordings, jacket spine information, logos, various cover designs, embossed serial numbers and a variety of other types depending upon particular recordings.

Referring again to Figure 8.1, the Audio-Header information in the above example includes the structured records about the sound recordings and the Audio-Body information includes the digitized sound. These data and their structure can be stored in the database and at the IRDS database and schema levels respectively. The image data can be handled in a manner similar to that described in the previous section.

There are other kinds of data involved as indicated in Figure 8.1. For instance, many sound recordings may exist concerning the same composition only performed by different artists. A researcher doing comparative analysis of the recordings may want to make voice annotations about his/her results or may reach certain conclusions at a more abstract level. The structure of some of this information could be stored at a higher level of the IRDS with other information requiring a level beyond the current IRDS levels.

In the example given above the original recordings are largely in analog form. In order to produce a digitized sound recording various sampling techniques can be used. The greater the rate the better the quality of the reproduced sound. As a point of reference, low quality sound may require only 20 KBytes/minute while high quality sound would require on the order of 0.5 MBytes/minute. As with imagery, the analog or digital recording is subject to analysis and therefore additional information may be generated. That is, through the analysis of the sound, textual data representing conclusions, knowledge, etc could be generated and thus produces additional input to the IRDS. Also, new representations of the sound could be generated and thus have to be stored in the database and thus affect the IRDS. Also, one can imagine a compressed version of some audio that contains key characteristics that could be used for index generation. These would then produce sound indexes that the IRDS might not be equipped to handle.

8.2.3 Video

The last type of media that we will discuss is video. As shown in Figure 8.1 a unit (tape, clip, etc) of video is described by header information as indicated by Video-Header. The amount of header information will vary depending upon how much is needed to describe the video. The actual video will probably be stored externally to the system since digital video consumes enormous amounts of secondary storage. For instance, 24 hours of digitized full motion video of TV quality without compression would take on the order of 2.5 terabytes of storage. However, one may be interested in storing short video chips, compressed video or swatches of video for a variety of applications. In this case the actual video would appear in a database with header information at the database or schema level of the IRDS.

The actual video can be viewed as a series of frames with each frame having an individual number. In this case each frame can be considered in the same way as image data except that there is considerable redundancy from frame to frame. Because of this redundancy, it may be best to consider groups of frames as a basic unit.

As with other types of multimedia a variety of other types of information can be extracted from the video. As with image data they can occur at all levels of the IRDS as well as levels that do not currently exist within the IRDS.

8.3 Extending the IRDS

It appears feasible to extend the IRDS to handle some types of multimedia data. However, before embarking on such a task, considerable additional study should be undertaken. This study should consider the particular characteristics of the various multimedia data types as well as the new types of information that they bring to the field. At that point one may view the task as designing a new IRDS to handle multimedia data as well as text and numeric data; rather than extending the current IRDS. We suggest that object-oriented techniques be used to encapsulate these multimedia types into object-types with well-defined interfaces and with methods to access and manipulate such type. These capabilities do not exist in the current IRDS specification.

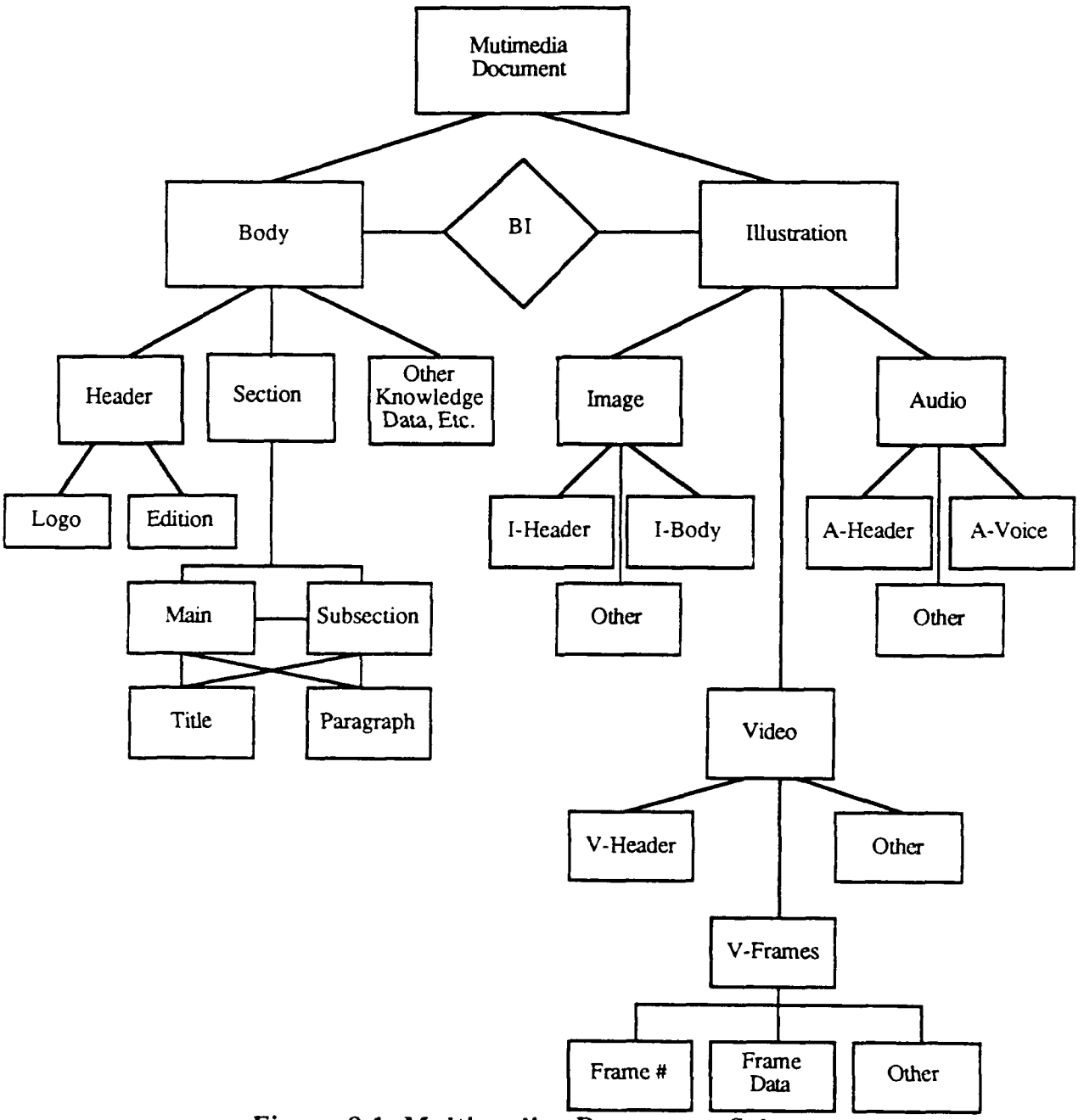


Figure 8.1 Multimedia Document Schema

9 Conclusions

This report has presented the results of our research into the concepts, tools, techniques and architectures for the Active Data/Knowledge Dictionary. We have studied the Information Resource Dictionary System as a candidate for the ADKD; although it supports a semantic data model, Chen's Entity/Relationship model, it will not support a fully object-oriented model which we feel is necessary for the ADKD.

In Section 3 of the report we focussed on a functional architecture for the ADKD. It should be viewed as an *environment* for the specification, development, and evolution of data-based and knowledge-based applications. The ADKD must provide a collection of services including: 1) User interface services for object-oriented browsing of complex data/knowledge organizations; 2) Data and knowledge acquisition services that support well-known knowledge representation paradigms such as productions, cases, and frame-based schemes as well semantic and object-oriented data models. It is clear that the ADKD must be able to represent behavioral or procedural knowledge in a declarative fashion so as to be able to reason about it; 3) Reasoning and knowledge management services to provide support to Data and Knowledge Engineers in their tasks, 4) Knowledge organization services to provide indexes into the object-oriented specifications of data/knowledge for efficient storage, retrieval and update; 5) Translation and mapping services to be able to communicate with existing systems, and 6) Learning and adaptation services to allow the data/knowledge schemata to evolve so as to reflect the reality of their application environments.

Section 4 examines the major proposals for query processing in heterogeneous databases and presents a novel approach, the Intelligent Heterogeneous Autonomous Database (InHead) architecture which incorporates a blackboard control model, multiple Knowledge Sources (KSs) that capture the structure and semantics of objects in the local databases, and an intelligent thesaurus that resolves semantic ambiguities of terms used in the local databases

Section 5 further explores the use of multiple knowledge sources and heterogeneous data and knowledge representations to solve problems. The concept of compiled knowledge resulting from a problem-solving episode is used to show how the various knowledge sources can evolve and improve the quality of their knowledge.

Section 6 presents our approach to schema evolution in object-oriented database systems. Previous approaches have solved portions of the problem, and our approach, which couples an ADKD with an AI truth maintenance system is original.

Section 7 shows how declaratively-defined rules can be integrated into an object-oriented data model. These rules can represent both integrity constraints and active productions. The concept of a rule schema is used to show how rules might be fired and which objects would be affected. This approach could be used to build a tool for Constraint and Rule Management for the ADKD.

Finally, section 8 presents our findings concerning multimedia types such as voice, video, and images within the ADKD. We discuss how the ADKD might be extended to include such types.

The concept of the Active Data/Knowledge Dictionary as an environment to support the analysis, design, development and management of object-oriented data- and knowledge-based applications is crucial to its acceptance within development organizations, and further research and development efforts are needed.

Bibliography

[A+87] R. Alonso et al. Concurrency control and recovery for global procedures in federated database systems. *Quarterly Bulletin of the IEEE Technical Committee on Data Engineering*, September 1987.

[Ahl84] M. Ahlsen. An architecture for object-management in OIS. *ACM Transactions on Office Information Systems*, 1984.

[Aik84] J. Aikens. A representation scheme using both frames and rules. In B. Buchanan and E. Shortliffe, editors, *Rule-Based Expert Systems*, pages 424–440. Addison-Wesley Publ. Co., Inc., Reading, MA, 1984.

[AK89] Serge Abiteboul and Paris C. Kanellakis. Object identity as a query language primitive. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 159–173, Portland, Oregon, 1989.

[All83] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[ALM82] F. W. Allen, M. E. S. Loomis, and M. V. Mannino. The integrated dictionary/ directory system. *ACM Computing Surveys*, 14(2):245–286, June 1982.

[AN86] S. Addanki and A. Nigam. KL-DB: Towards an unified approach to knowledge. In R. A. Meersman and A. C. Sernadas, editors, *Proc. of the Second IFIP 2.6 Working Conf. on Database Semantics, 'Data and Knowledge' (DS-2)*, pages 1–16, Netherlands, 1986. North Holland.

[And86] Peter Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Academic Press, Orland, FL, 1986.

[And87] T. Andrews. Combining language and database advances in an object-oriented development environment. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 430–440, October 1987.

[AP87] Krzysztof R. Apt and Jean-Marc Pugin. Maintenance of stratified databases viewed as a belief revision system. In *PODS '87*, pages 136–145, New York, 1987. ACM Press.

[AS89] B. Alpern and F. B. Schneider. Verifying temporal properties without temporal logic. *ACM Trans. on Programming Languages and Systems*, 11(1):147–167, January 1989.

[B+81] P. A. Bernstein et al. Query processing in a system for distributed databases SDD-1. *ACM Transactions on Database Systems*, 6(4):602–605, December 1981.

[B+86] C. Batini et al. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–363, December 1986.

[B+87a] Jay Banerjee et al. Data model issues for object-oriented applications. *ACM Transactions on Office Information Systems*, 5(1):3–26, January 1987.

[B+87b] Jay Banerjee et al. Semantics and implementation of schema evolution in object-oriented databases. In *SIGMOD '87*, pages 311–322, New York, 1987. ACM Press.

[B+87c] S. Bhalla et al. A technical comparison of distributed heterogeneous database management systems. In A. Gupta and S. Madnick, editors, *Integrating Distributed Homogeneous and Heterogeneous Databases - Prototypes, Vol 3*, pages 159–218. MIT Press, Cambridge, MA, 1987.

[B+89] Robert Bretl et al. The GemStone data management system. In Won Kim and Frederick H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, pages 283–308. ACM Press, New York, 1989.

[Ban87a] J. Banerjee. Data model issues for object-oriented applications. *ACM Transactions on Office Information Systems*, pages 3–26, January 1987.

[Ban87b] J. Banerjee. Semantics and implementation of schema evolution in object-oriented databases. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 1987.

[BBG80] C. Bernstein, B. Blaustein, and N. Goodman. Fast maintenance of integrity assertions using redundant aggregate. In *Proceedings of the 6th Int'l Conf. on Very Large Databases*, pages 126–136, Alfonso Cardenas, 1980.

[BCG+87] J. Banerjee, H. Chou, J. F. Garza, W. Kim, and D. Woelk. Data model issues for object-oriented applications. *ACM Transactions on Office Information Systems*, 5:3–26, 1987.

[Bee87] D. Beech. Groundwork for an object database model. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*. The MIT Press, Cambridge, MA, 1987.

[BG88] Alan H. Bond and Les Gasser. An analysis of problems and research in DAI. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 3–36, San Mateo, CA, 1988. Morgan Kaufmann.

[BG89] M. Burgin and V. Gladun. Mathematical foundations of semantic networks theory. In J. Demetrovics B. Thalheim, editor, *Proc. 2nd Symposium on Mathematical Fundamentals of Databases Systems*, pages 117–135, Visegrad, Hungary, 1989. Springer-Verlag.

[BH89] Anders Björnerstedt and Christer Hultén. Version control in an object-oriented architecture. In Won Kim and Frederick H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, pages 451–485. ACM Press, New York, 1989.

[BK85] D. Batory and W. Kim. Modeling concepts for VLSI CAD objects. *ACM Transactions on Database Systems*, 10(3), September 1985.

[BKK85] F. Bancilhon, W. Kim, and H. Korth. A model of CAD transactions. *Proc. Intl. Conf. on Very Large Data Bases*, pages 12–21, August 1985.

[BKK88] J. Banerjee, W. Kim, and K. C. Kim. Queries in object-oriented databases. *Proc. 4th Intl. Conf. on Data Engineering*, February 1988.

[Bla86] A. Black. Object structure in the Emerald system. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 67–77, 1986.

[BM88a] David Beech and Brom Mahbod. Generalized version control in an object-oriented database. In *DE '88*, pages 14–22, Washington, D.C., 1988. IEEE Computer Society.

[BM88b] E. Bertino and D. Musto. Correctness of semantic integrity checking in database management systems. *Acta Informatica*, 26:25–57, 1988.

[BM89] M. L. Brodie and F. Manola. Database management: A survey. In J. Mylopolous and M. Brodie, editors, *Readings in Artificial Intelligence and Databases*. Morgan-Kaufmann, Palo Alto, CA, 1989.

[Boc89] Jorge B. Bocca. Rule for implementing very large knowledge base systems. *ACM SIGMOD Record*, 18(3):29–35, September 1989.

[BR87] B. R. Badrinath and Krithi Ramamritham. Semantics-based concurrency control: Beyond commutativity. In *Proc. 3rd Intl. Conf. on Data Engineering*, pages 304–311, California, 1987.

[Bre85] M. Breitbart. ADDS: Another multidatabase management system. In *Distributed Data Sharing Systems*, pages 7–24. North Holland Pub. Co., Amsterdam, Netherlands, 1985.

[Bro89] M. L. Brodie. Future intelligent information systems: AI and database technologies working together. In J. Mylopolous and M. Brodie, editors, *Readings in Artificial Intelligence and Databases*. Morgan-Kaufmann, Palo Alto, CA, 1989.

[BS83] D. G. Bobwor and M. Stefik. *The LOOPS Manual*. Xerox Corporation, Palo Alto, CA., 1983.

[BS88a] A. Bjornerstedt and B. Stefan. ADVANCE: An object management system. In *Proc. 3rd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 206–221. ACM, 1988.

[BS88b] Y. Breitbart and A. Silberschatz. Multidatabase update issues. In *SIGMOD '88*, pages 135–142, 1988.

[BZ87] T. Bloom and S. B. Zdonik. Issues in the design of object-oriented database programming languages. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 441–451, 1987.

[Cap87] M. Caplinger. An information system based on distributed objects. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 126–137, 1987.

[Car87] A. F. Cardenas. Heterogeneous distributed database management: The HD-DBMS. *Proceedings of the IEEE*, 75(5), May 1987.

[CD87] L. Cholvy and Robert Demolombe. Querying a rule base. In *Expert Database Systems: Proceedings of the First International Conference*, Menlo Park, CA, 1987. The Benjamin/Cummings Publishing Company, Inc.

[CF85] M. Casanova and A. Furtado. A family of temporal languages for the description of transition constraints. In H. Gallaire, J. Minker, and J. M. Nicolas, editors, *Advances in Database Theory*, New York, 1985. Plenum Press.

[CGM90] Upen S. Chakravarthy, John Grant, and Jack Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2), June 1990.

[Cha82] J. M. Chang. A heuristic approach to distributed query processing. In *VLDB 82*, 1982.

[Cha89] Upen S. Chakravarthy. Rule management and evaluation: An active DBMS perspective. *ACM SIGMOD Record*, 18(3):20–28, September 1989.

[Che76] Peter P.S. Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[Chr84] S. Christodoulakis. Implications of certain assumptions in database performance evaluation. *ACM Transactions on Database Systems*, 9(2):163–186, June 1984.

[CK86] H. T. Chou and W. Kim. A unifying framework for versions in a CAD environment. In *VLDB '86*, 1986.

[CK88] H. T. Chou and W. Kim. Versions and change notification in an object-oriented database system. In *DAC '88*, 1988.

[Coh89] Donald Cohen. Compiling complex database transition triggers. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 225—234, Portland, Oregon, 1989.

[Cox86] B. J. Cox. *Object Oriented Programming: An Evolutionary Approach.* Addison-Wesley, Reading, MA, 1986.

[Cox87] B. Cox. Message/Object Programming: An evolutionary change in programming technology. In G. E. Peterson, editor, *Tutorial: Object-Oriented Computing, Concepts*, pages 150–161. The Computer Society Press of the IEEE, 1987.

[CW83] James Clifford and David W. Warren. Formal semantics for time in databases. *ACM Transactions on Database Systems*, 8(2):214–254, June 1983.

[D*85] S. M. Deen et al. The architecture of a generalised distributed database system - PRECI*. *Computer Journal*, 28(3):282–290, 1985.

[D*87] U. Dayal et al. Simplifying complex objects: The PROBE approach to modelling and querying them. In *BTW-87 (Datenbanksysteme in Buro, Technik und Wissenschaft*, Damnstadt, Germany, 1987.

[D+88] P. A. Dwyer et al. Answer: Army's nonprogrammer system for working encyclopedia requests. *Newsletter of the Computer Society of the IEEE Technical Committee on Distributed Processing*, 10(2):5–13, November 1988.

[Dat81] C. J. Date. *An Introduction to Database Systems, Third Edition.* Addison-Wesley Publ. Co., Inc., Reading, MA, 1981.

[Dav87] Ernest Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32:281–331, 1987.

[DBB+88] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin, D. McCarthy, A. Rosenthal, S. Sarin, M. J. Carey, M. Livny, and R. Jauhari. The HiPAC Project: Combining Active Databases and Timing Constraints. *ACM SIGMOD Record*, 17(1):51–70, March 1988.

[DBM88] Umeshwar Dayal, Alejandro P. Buchmann, and Dennis R. McCarthy. Rules are objects too: A knowledge model for an active, object-oriented database system. In G. Goos and J. Hartmanis, editors, *Proc. of 2nd Int'l Workshop on Object-Oriented Database Systems*, pages 129–143, FRG, 1988. Bad Munster am Stein-Ebernberg.

[DD86] Klaus Dittrich and Umesh Dayal, editors. *1986 International Workshop on Object-Oriented Database Systems*, New York, 1986. ACM Press.

[DD88] J. Duhl and C. Damon. A performance comparison of object and relational databases using the Sun benchmark. *Proc. 3rd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 153–163, 1988.

[Dec87] Hendrik Decker. Integrity enforcement on deductive databases. In Larry Kerschberg, editor, *Expert Database Systems*, pages 381–395. Morgan Kaufmann, 1987.

[DG88] Edward R. Dougherty and Charles R. Giardina. *Mathematical Methods for Artificial Intelligence and Autonomous Systems.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1988.

[DH84] U. Dayal and H. Hwang. View definition and generalization for database integration in a multidatabase system. *IEEE Transactions on Software Engineering*, SE-10(6):628–645, November 1984.

[Dit86] K. R. Dittrich. Object-oriented database systems: the notion and the issues. In K. Dittrich and U. Dayal, editors, *Proceedings of the 1986 International Workshop on Object-Oriented Database Systems*, pages 2–4, 1986.

[dK86a] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127–162, March 1986.

[dK86b] Johan de Kleer. Extending the ATMS. *Artificial Intelligence*, 28(2):163–196, March 1986.

[dKF90] Johan de Kleer and Kenneth D. Forbus. Truth maintenance systems. Tutorial notes from the Eighth National Conference on Artificial Intelligence, July 1990.

[DO85] G. B. Davis and M. H. Olson. *Management Information Systems: Conceptual Foundations, Structure, and Development.* McGraw-Hill, New York, 1985.

[Dol88] D. R. Dolk. Model management and structured modeling: The role of an information resource dictionary system. *Communications of the ACM*, 31(6):704–718, June 1988.

[Doy79] Jon Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3):231–272, November 1979.

[DP88] R. Dechter and J. Pearl. Network-based heuristics of constraint-satisfaction. *Artificial Intelligence*, 34:1–38, 1988.

[DPW87] S. De, S. Pan, and A. Whinston. Temporal semantics and natural language processing in a decision support system. *Information Systems*, 12:29–47, 1987.

[Duy82] J. Van Duyn. *Developing a Data Dictionary System.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[DWE89] L. Delcambre, J. Waramahaputi, and J. Etheridge. Pattern match reduction for the relational production language in the USL MMDBS. *ACM SIGMOD Record*, 18(3), September 1989.

[Dwy81] B. Dwyer. A user friendly algorithm. *Communications of the ACM*, pages 556–561, September 1981.

[E+87] Denise J. Ecklund et al. DVSS: a distributed version storage server for CAD applications. In *VLDB '87*, pages 443–454, Palo Alto, CA, 1987. Morgan-Kaufmann.

[EG89] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 399–407, Portland, Oregon, 1989.

[Eli87] Alan L. Eliason. *Systems Development.* Little, Brown and Company, 1987.

[Elm87] A. K. Elmagarmid. When will we have true heterogeneous databases? In *Proceedings of the Second Fall Joint Computer Conference, Dallas, Texas*, page 746, Washington, D.C., 1987. IEEE Computer Society.

[ET88] Denise J. Ecklund and Fred Tonge. A context mechanism to control sharing in a design database. In *DAC '88*, pages 344–350, Washington, D.C., 1988. IEEE Computer Society.

[F+88] Tim Finin et al. Adding forward chaining and truth maintenance to Prolog. Technical Report PRC-LBS-8802, UNISYS Paoli Research Center, Paoli, PA, September 1988.

[FG86] Elizabeth Fong and Alan H. Goldfine. Database directions: Information resource management-making it work. Technical report, National Bureau of Standards, U.S. Government Printing Office, Washington D.C., 1986.

[Fia88] J. Fiadeiro. Specification and verification of database dynamics. *Acta Informatica*, 25:625–661, 1988.

[Fil88] Robert Filman. Reasoning with worlds and truth maintenance in a knowledge-based programming environment. *Communications of the ACM*, 31(4):382–401, April 1988.

[Fis87] D. Fishman. IRIS: an object-oriented database management system. *ACM Transactions on Office Information Systems*, 5(1):48–69, January 1987.

[Fis89] D. Fishman. Overview of the IRIS DBMS. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*. Addison-Wesley Publ. Co., Inc., Reading, MA, 1989.

[FR88] E. N. Fong and B. K. Rosen. Guide to distributed database management. Technical Report Special Publication 500-154, National Bureau of Standards, Gaithersburg, Maryland, April 1988.

[Fra87] M. C. Frame. *Automatic Translation of Query Languages in Heterogeneous Database Management Systems*. PhD thesis, The School of Engineering and Applied Science, The George Washington University, 1987.

[FS86] J. Fiadeiro and A. Sernadas. The INFOLOG linear tense propositional logic of events and transactions. *Information Systems*, 11:61–85, 1986.

[G+84] V. Gligor et al. Interconnecting heterogeneous database management systems. *IEEE Computer*, pages 33–43, June 1984.

[G+86] V. Gligor et al. Transaction management in distributed heterogeneous database management systems. *Information Systems*, 11(4):287–297, 1986.

[GCHR87] A. Garvey, C. Cornelius, and B. Hayes-Roth. Computational costs versus benefits of control reasoning. Technical Report KSL-87-11, Stanford University, 1987.

[GK87] J. F. Garza and W. Kim. Transaction management in an object-oriented database system. Technical Report ACA-ST-292-87, Microelectronics and Computer Technology Corporation, Austin, Texas, September 1987.

[GK88] A. Goldfine and Patricia Konig. A technical overview of the Information Resource Dictionary System. Technical report, National Bureau of Standards, Washington, DC, 1988.

[GM78] H. Gallaire and J. Minker. *Logic and Databases*. Plenum Press, New York, 1978.

[GM83] Hector Garcia-Molina. Using semantic knowledge for transaction processing in a distributed database. *ACM Transactions on Database Systems*, 8(2):186–213, June 1983.

[Gol82] Alan H. Goldfine. Database directions: Information resource management strategies and tools. Technical report, National Bureau of Standards, Washington, DC, 1982.

[GPZ85] V. Gligor and R. Popescu-Zeletin. Concurrency control issues in distributed heterogeneous database management systems. In *Distributed Data Sharing Systems*, pages 43–56. North Holland Pub. Co., Amsterdam, Netherlands, 1985.

[Gup89] Amar Gupta. *Integration of Information Systems: Bridging Hetereogeneous Databases*. IEEE Computer Society, Washington, D.C., 1989.

[GW88] S. K. Goyal and R. W. Worrest. Expert system applications to network management. In J. Liebowitz, editor, *Expert System Applications to Telecommunications*, pages 3–44. John Wiley & Sons, Inc., New York, NY, 1988.

[Han89] Eric N. Hanson. An initial report on the design of Ariel: A DBMS with an integrated production rule system. *ACM SIGMOD Record*, 18(3):12–19, September 1989.

[HB85] G. F. Hoffnagle and W. E. Beragi. Automating the software development process. *IBM Systems Journal*, 24(2), 1985.

[Hei87] S. Heiler. Heterogeneous databases: Some outstanding problems. In *Proceedings of the Second Fall Joint Computer Conference*, pages 749–750, Washington, D.C., 1987. IEEE Computer Society.

[Her87] M. P. Herlihy. Optimistic concurrency control for abstract data types. *Operating Systems Review*, 21(2), April 1987.

[HHR87] M. Hewett and B. Hayes-Roth. The BB1 architecture: A software engineering view. Technical report, Stanford University, Palo Alto, CA, 1987.

[HK86] S. E. Hudson and R. King. CACTIS: A database system for specifying functionally-defined data. In K. Dittrich and U. Dayal, editors, *Proceedings of the 1986 International Workshop on Object-Oriented Database Systems*, pages 26–37, Pacific Grove, California, 1986.

[HK87] Richard Hull and Roger King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[HM85] D. Heimbigner and D. McLeod. A federated architecture for information management. *ACM Transactions on Office Information Systems*, 3(3):253–278, July 1985.

[HN87] Brent Hailpern and Van Nguyen. A model for object-based inheritance. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 147–164. MIT Press, Cambridge, MA, 1987.

[HR85] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence Journal*, 26:251–321, 1985.

[HR+87a] B. Hayes-Roth et al. A modular and layered environment for reasoning about action. Technical Report Technical Report KSL-86-38, Stanford University, Palo Alto, CA, 1987.

[HR87b] H. C. Howard and D. R. Rehak. KADBASE – a prototype expert system-database for integrated CAI environments. In *Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87*, pages 804–808, Seattle, Washington, 1987. American Association for Artificial Intelligence.

[HR89] H. C. Howard and D. R. Rehak. KADBASE – a prototype expert system-database interface for engineering systems. *IEEE Expert*, Spring, 1989.

[HRH88] B. Hayes-Roth and M. Hewett. BB1: An implementation of the blackboard control architecture. In R. Englemore and T. Morgan, editors, *Blackboard Systems*, pages 297–313. Addison-Wesley Publ. Co., Inc., Reading, MA, 1988.

[HS89] Richard Hull and Jianwen Su. On accessing object-oriented databases: Expressive power, complexity, and restrictions. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 147–158, Portland, Oregon, 1989.

[HSC86] M. T. Harandi, T. Schang, and S. Cohen. Rule base management using meta knowledge. *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 261–267, 1986.

[HTY89] Richard Hull, Katsumi Tanaka, and Masatoshi Yoshikawa. Behavior analysis of object-oriented databases: Method structure, execution trees, and reachability. In W. Litwin and H.-J. Schek, editors, *Third International Conference FODO*, pages 372–388. Springer-Verlag Lecture Notes in Computer Science, 1989.

[HW88] M. P. Herlihy and W. E. Weihl. Hybrid concurrency control for abstract data types. *Proc. of the ACM symposium on Principles of Database Systems*, 1988.

[HY79] A. Hevner and S. B. Yao. Query processing in distributed database systems. *IEEE Transactions on Software Engineering*, SE-5:177–187, May 1979.

[Imi87] Tomasz Imielinski. Intelligent query answering in rule based systems. *J. Logic Programming*, 4:229–257, 1987.

[Inm89a] W.H. Inmon. *Advanced Topics in Information Engineering*. QED Information Sciences, MA, 1989.

[Inm89b] W.H. Inmon. *Data Architecture: The Information Paradigm*. QED Information Sciences, MA, 1989.

[Jar86] M. Jarke. Control of search and knowledge acquisition in large-scale KBMS. In *On Knowledge Based Management Systems*, pages 507–526. Springer-Verlag, Berlin, 1986.

[JHR86] M. Johnson and B. Hayes-Roth. Integrating diverse reasoning methods in the BB1 blackboard control architecture. Technical Report KSL 86-76, Stanford University, Palo Alto, CA, 1986.

[JJ89] Matthias Jarke and Manfred Jeusfeld. Rule representation and management in conceptbase. *ACM SIGMOD Record*, 18(3):46–51, September 1989.

[JK84] M. Jarke and J. Koch. Query optimization in database systems. *ACM Computing Surveys*, 16(2):111–152, June 1984.

[JR86] M. B. Jones and R. F. Rashid. Mach and Matchmaker: kernel and language support for object-oriented distributed systems. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 67–77, 1986.

[K+83] L. Kerschberg et al. Information system integration: A metadata management approach. In *Proceedings of the Fourth International Conference on Information Systems*, pages 223–239, Houston, TX, 1983.

[K+86] Randy H. Katz et al. Version modeling concepts for computer-aided design databases. In *SIGMOD '86*, pages 379–386, New York, 1986. ACM Press.

[K+87] Won Kim et al. Enhancing the object-oriented concepts for database support. In *DE '87*, pages 291–292, Washington, D.C., 1987. IEEE Computer Society.

[K+89] Won Kim et al. Features of the ORION object-oriented database system. In Won Kim and Frederick H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, pages 251–282. ACM Press, New York, 1989.

[KB88] M. Rajini Kanth and Prasanta K. Bose. Extending an assumption-based truth maintenance system to databases. In *DE '88*, pages 354–359, Washington, D.C., 1988. IEEE Computer Society.

[KBD+90] L. Kerschberg, R. Baum, K. DeJong, A. Waisanen, J. Yoon, I. Huang, K. Eisgruber, and B. Utz. Knowledge and data engineering of a telecommunications network. In *Proceedings of the 9th International Conference on Entity-Relationship Approach*, Lausanne, Switzerland, 1990.

[KBG89] Won Kim, Elisa Bertino, and Jorge F. Garza. Composite object revisited. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 337–347, Portland, Oregon, 1989.

[KBH89] L. Kerschberg, R. Baum, and J. Hung. KORTEX: An expert database system shell for a knowledge-based entity relationship model. In *Proceedings of the 8th International Conference on Entity-Relationship Approach*, Toronto, Canada, 1989.

[KC87] R. H. Katz and E. Chang. Managing change in a computer-aided design database. In *VLDB '87*, pages 455–462, Palo Alto, CA, 1987. Morgan-Kaufmann.

[KC88] W. Kim and H. T. Chou. Versions of schema for object-oriented databases. In *VLDB '88*, pages 148–159, Palo Alto, CA, 1988. Morgan-Kaufmann.

[KdMS89] J. Kiernan, C. de Maindreville, and E. Simon. The design and implementation of an extendible deductive database system. *ACM SIGMOD Record*, 18(3):68–77, September 1989.

[Kim87] W. Kim. Composite object support in an object-oriented database system. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, October 1987.

[Kim89] W. Kim. A model of queries for object-oriented databases. *Proc. Intl. Conf. on Very Large Data Bases*, August 1989.

[Kim90] Won Kim. Research directions in object-oriented databases. Technical Report ACT-OODS-013-90, Microelectronics and Computer Technology Corporation, Austin, TX, January 1990.

[Kin86] R. King. A database management system based on an object-oriented model. In L. Kerschberg, editor, *Expert Database Systems, Proc. from the First Intl. Workshop*, pages 443–468. Benjamin/Cummings, 1986.

[KKD89] W. Kim, K. C. Kim, and A. Dale. Indexing techniques for object-oriented databases. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*. Addison-Wesley Publ. Co., Inc., Reading, MA, 1989.

[KL89a] Michael Kifer and Georg Lausen. F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 134–146, Portland, Oregon, 1989.

[KL89b] Won Kim and Frederick H. Lochovsky, editors. *Object-Oriented Concepts, Databases, and Applications*. ACM Press, Addison-Wesley Publishing Co., New York, 1989.

[KM84a] R. King and D. McLeod. A unified model and methodology for conceptual database design. In M. Brodie, J. Mylopoulos, and J. Schmidt, editors, *On Conceptual Modelling: Perspectives From Artificial Intelligence, Databases, and Programming Languages*, New York, 1984. Springer-Verlag.

[KM84b] Roger King and Dennis McLeod. A unified model and methodology for conceptual database design. In M. Brodie, J. Mylopoulos, and J. Schmidt, editors, *On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*. Springer-Verlag, Berlin, 1984.

[KMK89] K. Kaufman, R. S. Michalski, and L. Kerschberg. Mining for knowledge in data: Goals and general description of the INLEN system. In G. Piatetski-Shapiro, editor, *IJCAI-89 Workshop on Knowledge Discovery in Databases*, Mountain View, CA, 1989. American Association for Artificial Intelligence.

[KN86] S.N. Khoshafian and N.Setrag. Object Identity. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 406–416, 1986.

[Kow78] R. Kowalski. Logic for data description. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 77–103, New York, 1978. Plenum Press.

[KP76] L. Kerschberg and J. E. S. Pacheco. A functional data base model. Technical report, Portificia Univ. Catolica do Rio De Janeiro, Rio de Janeiro, Brazil, 1976.

[KSS87] R. Kowalski, F. Sadri, and P. Soper. Integrity checking in deductive databases. In *Proc. 13th Intl. Conf. on Very Large Data Bases*, pages 61–69, Brighton, 1987.

[L+84] J. A. Larson et al. An interface for novice and infrequent database system users. In *National Computer Conference*, pages 523–529, Reston, VA, 1984. AFIPS Press.

[L+86] W. Litwin et al. Multidatabase interoperability. *Computer*, 19, December 1986.

[LA87] W. Litwin and A. Abdellatif. An overview of the multi-database manipulation language MDSL. *Proceedings of the IEEE*, 75(5), May 1987.

[LB85] H. J. Levesque and R. J. Brachman. A fundamental tradeoff in knowledge representation and reasoning. In R. Brachman and H. Levesque, editors, *Readings in Knowledge Representation*, pages 41–70. Morgan Kaufmann, Los Altos, 1985.

[LB86] D. Libes and E. Barkmeyer. IMDAS - an overview. Technical report, National Bureau of Standards, Gaithersburg, MD, 1986.

[Lee88] Ronald M. Lee. Logic, semantics and data modeling: An ontology. In R. A. Meersman and A. C. Sernadas, editors, *Proc. of the Second IFIP 2.6 Working Conf. on Database Semantics, 'Data and Knowledge' (DS-2)*, pages 221–244, Netherland, 1988. North-Holland.

[Len87] Maurizio Lenzerini. Covering and disjointness constraints in type networks. In *DE '87*, pages 386–393, Washington, D.C., 1987. IEEE Computer Society.

[Lit85] W. Litwin. An overview of the multidatabase system MRDSM. In *Proceedings of the ACM National Conference*, Denver, Colorado, 1985.

[Lit87] W. Litwin. The future of heterogeneous databases. In *Proceedings of the Second Fall Joint Computer Conference*, pages 751–752, Dallas, TX, 1987.

[LJ88] N. A. Lorentzos and G. Johnson. Extending relational algebra to manipulate temporal data. *Information Systems*, 13:289–296, 1988.

[LKCS90] Aaron Larson, John Kimball, Jeff Clark, and Bob Schrag. KBSA framework. Technical Report RADC-TR-90-349, Honeywell, Inc. Systems & Research Center, 3660 Technology Drive, Minneapolis, MN 55418, December 1990.

[LM88a] Qing Li and Dennis McLeod. Object flavor evolution in an object-oriented database system. In *Proceedings of the Conference on Office Information Systems*, pages 265–275, New York, 1988. ACM Press.

[LM88b] Qing Li and Dennis McLeod. Object flavor evolution through learning in an object-oriented database system. In *EDS '88*, pages 241–256, Fairfax, VA, 1988. George Mason Foundation.

[LR89] W. Litwin and N. Roussopoulos. Interoperability of multiple autonomous databases. Technical Report SRC TR 89-12, Systems Research Center, University of Maryland, College Park, MD, 1989.

[LS87] U. Lipeck and G. Saake. Monitoring dynamic integrity constraints based on temporal logic. *Information Systems*, 12:255–269, 1987.

[LSL83] H. Lefkovits, E. H. Sibley, and S. Lefkovits. *Information Resource/Data Dictionary Systems*. QED Information Sciences, MA, 1983.

[LZ88] W. Litwin and A. Zeroual. Advances in multidatabase systems. In R. Speth, editor, *Research into Networks and Distributed Applications*, pages 1137–1151. North Holland Pub. Co., Amsterdam, Netherlands, 1988.

[M+88] M. Morgenstern et al. Constraint-based systems: Knowledge about data. In L. Kerschberg, editor, *Expert Database Systems: Proceedings from the Second International Conference*, pages 23–43, Menlo Park, CA, 1988. The Benjamin/Cummings Publishing Company, Inc.

[Mai86] D. Maier. Development of an object-oriented DBMS. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 472–482, 1986.

[Mai89] D. Maier. Making database systems fast enough for CAD applications. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*. Addison-Wesley Publ. Co., Inc., Reading, MA, 1989.

[MB88] B. Martens and M. Bruynooghe. Integrity constraint checking in deductive databases. In Larry Kerschberg, editor, *Expert Database Systems*, pages 297–310. The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA, 1988.

[MB90] D. Miranker and D. Brant. An algorithmic basis for active databases. In *Proc.of the Intl. Conf. on Data Engineering*, Washington, D.C., 1990. IEEE Computer Society.

[McC88] J. L. McCarthy. Information systems design for material properties data. Technical report, Lawrence Berkeley Laboratory, Computer Science Department, March 1988.

[McC89] Dennis R. McCarthy. The architecture of an active data base management system. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 215–224, Portland, Oregon, 1989.

[MD89] D. McCarthy and U. Dayal. The architecture of an active data base management system. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, New York, 1989. ACM Press.

[Mey88] Bertrand Meyer. *Object-oriented software construction*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1988.

[MH86] R. Mohr and T. Henderson. Arc and path consistency. *Artificial Intelligence*, 28:225–233, 1986.

[Mic88] J. Micallef. Encapsulation, reusability, and extensibility in object-oriented programming languages. *Journal of Object-Oriented Programming*, 1(1):12–36, April 1988.

[Min87] N. H. Minsky. A law-based approach to object-oriented programming. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 482–492, 1987.

[ML87] T. Merrow and J. Laursen. A pragmatic system for shared objects. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 103–110, 1987.

[Mor81] W. Morris. *The American Heritage Dictionary of the English Language*. Houghton Mifflin, Boston, MA, 1981.

[Mor86] M. Morgenstern. The role of constraints in databases, expert systems and knowledge representation. In Larry Kerschberg, editor, *Expert Database Systems: Proceeding of the First International Workshop*, pages 351–368, Menlo Park, CA, 1986. The Benjamin/Cummings Publishing Company, Inc.

[Mor89] Matthew Morgenstern. Intelligent database systems. Technical Report Final report A003, SRI Project 2201, SRI International, Menlo Park, CA, October 1989.

[Mot86] A. Motro. Query generalization: A method for interpreting null answers. In L. Kerschberg, editor, *Expert Database Systems: Proceedings of the First International Workshop*, pages 597–616, Menlo Park, CA, 1986. The Benjamin/Cummings Publishing Company, Inc.

[Mot87] A. Motro. Superviews: virtual integration of multiple databases. *IEEE Transactions on Software Engineering*, 13:785–798, 1987.

[Mot89] A. Motro. Using integrity constraints to provide intensional answer to relational queries. In *The Fifteenth Int'l Conf. on VLDB*, pages 237–246, Amsterdam, 1989.

[MS87] David Maier and Jacob Stein. Development and implementation of an object-oriented DBMS. In Bruce Shriver and Peter Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 355–392. MIT Press, Cambridge, MA, 1987.

[MW84] Z. Manna and P. Wolper. Synthesis of communicating processes for temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 6:68–93, 1984.

[NB88] K. Narayanaswamy and K. V. Bapa Rao. An incremental mechanism for schema evolution in engineering domains. In *DE '88*, pages 294–301, Washington, D.C., 1988. IEEE Computer Society.

[Ngu86] G. T. Nguyen. Object prototypes and database samples for expert database systems. In Larry Kerschberg, editor, *the First International Conf. on Expert Database Systems*, pages 3–14, Charleston, 1986.

[Nie87] O.M. Nierstratz. Active objects in Hybrid. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 243–253, 1987.

[Nii86] H. P. Nii. Blackboard systems. *AI Magazine*, 7(3):38–53, 1986.

[NK86] S. B. Navathe and L. Kerschberg. Role of data dictionaries in information resource management. *Information and Management*, 10:21–46, 1986.

[NS87] K. Narayanaswamy and Walt Scacchi. Maintaining configurations of evolving software systems. *IEEE Transactions on Software Engineering*, 13(3):324–334, March 1987.

[Obj90] Object-Oriented Database Task Group. Revised draft of reference model for object data management. Accredited Standards Committee Document Number: OODB 89-01R5, December 1990.

[OHK87] P. D. O'Brien, D. C. Halbert, and M. F. Kilian. The Trellis programming environment. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 91–102, 1987.

[Ont86] Ontologic, Inc. Vbase technical overview. Technical report, Ontologic, Inc., Billerica, Massachusetts, 1986.

[OS89] Fredy Oertly and Gerald Schiller. Evolutionary database design. In *DE '89*, pages 618–624, Washington, D.C., 1989. IEEE Computer Society.

[Ozk86] E. Ozkarahan. *Database Machines and Database Management.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986.

[P+89] Walter D. Potter et al. Model instantiation for query driven simulation in Active KDL. In *23rd Annual Simulation Symposium.* 1990 SCS Eastern Multiconference, 1989.

[Pap89] M. P. Papazoglou. A semi-decentralized architecture for the integration of distributed database systems. Technical report, German Research Center for Computer Science, GMD, Scholss Birlinghoven, Germany, 1989.

[Pau87] Benjamin D. Paul. Learning strategies by reasoning about rules. In *Proc. of the Tenth Int'l Joint Conf. on Artificial Intelligence*, pages 256–259, Milan, 1987.

[PCKW89] K. Parsaye, M. Chignell, S. Khoshafian, and H. Wong. *Intelligent Databases.* John Wiley & Sons, Inc., New York, NY, 1989.

[PK89] W. D. Potter and L. Kerschberg. A unified approach to modeling knowledge and data. In R. A. Meersman and A. C. Sernadas, editors, *Proc. of the Second IFIP 2.6 Working Conf. on Database Semantics, 'Data and Knowledge' (DS-2)*, pages 189–204, Netherlands, 1989. North-Holland.

[PM88] J. Peckham and F. Maryanski. Semantic data models. *ACM Computing Surveys*, 20(3):153–189, 1988.

[Pot87] Walter D. Potter. *A knowledge-based approach to enterprise modeling: the foundation.* PhD thesis, University of South Carolina, 1987.

[PS87] D. Jason Penney and Jacob Stein. Class modification in the GemStone object-oriented DBMS. In *OOPSLA '87*, pages 111–117, New York, 1987. ACM Press.

[PT88] W. D. Potter and R. P. Trueblood. Traditional, semantic, and hyper-semantic approaches to data modeling. *IEEE Computer*, pages 53–63, June 1988.

[QS87] X. Qian and D. R. Smoth. Integrity constraint reformulation for efficient validation. In *Proc. 13th Intl. Conf. on Very Large Data Bases*, pages 417–425, Brighton, 1987.

[R+88] M. Rusinkiewicz et al. Omnibase: Design and implementation of a multidatabase system. *Newsletter of the Computer Society of the IEEE Technical Committee on Distributed Processing*, 10(2):20–28, November 1988.

[Ram87] K. Ramamritham. Verification of resource controller processes. *Information Systems*, 12:57–67, 1987.

[RCB89] A. Rosenthal, S. Chakravarthy, and B. Blaustein. Situation monitoring for active databases. In *The Fifteenth Int'l Conf. on VLDB*, pages 455–464, Amsterdam, 1989.

[Rit86] J. Rit. Propagating temporal constraints for scheduling. In *American Association for Artificial Intelligence*, pages 383–388, Philadelphia, 1986.

[RS87] L. Rowe and M. Stonebraker. The POSTGRES data model. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 1987.

[RS89] C. Riesbeck and R. Schank. *Inside Case-based Reasoning*. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1989.

[Rus87] M. Rusinkiewicz. Heterogeneous databases - towards a federation of autonomous systems. In *Proceedings of the Second Fall Joint Computer Conference*, page 753, Washington, D.C., 1987. IEEE Computer Society.

[RWK88] F. Rabitti, D. Woelk, and W. Kim. A model of authorization for object-oriented and semantic databases. In *Proc. Intl Conf. on Extending Database Technology*, 1988.

[S+81] J. M. Smith et al. MULTIBASE – integrating heterogeneous distributed database systems. In *Proceedings of the 1981 National Computer Conference*, pages 487–499, Reston, VA, 1981. AFIPS Press.

[S+84] P. M. Stocker et al. PROTEUS: A heterogeneous distributed database project. In P. M. Stocker, P. M. Gray, and M. P. Atkinson, editors, *Databases - Role and Structure: An Advanced Course*. Cambridge University Press, Cambridge, England, 1984.

[S+85] W. Staniszkis et al. Network database management system architecture. In *Distributed Data Sharing Systems*, pages 57–76. North Holland Pub. Co., Amsterdam, Netherlands, 1985.

[S+87] R. D. Semmel et al. A functional architecture for an AI interface to the HQDA corporate database. Technical report, Applied Physics Laboratory, Johns Hopkins University, Baltimore, MD, June 1987.

[SB86] M. Stefik and D. G. Bobrow. Object-oriented programming: Themes and variations. *AI Magazine*, 6:40–62, 1986.

[Sch86] C. Schaffert. An introduction to Trellis/Owl. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, September 1986.

[She87] A. P. Sheth. When will we have true heterogeneous database systems? In *Proceedings of the Second Fall Joint Computer Conference*, pages 747–748, Washington, D.C., 1987. IEEE Computer Society.

[She88a] A. P. Sheth. Building federated database systems. *Newsletter of the Computer Society of the IEEE Technical Committee on Distributed Processing*, 10(2):50–58, November 1988.

[She88b] A. P. Sheth. A tool for integrating conceptual schemas and user views. In *Proc. 4th Intl. Conf. on Data Engineering*, Washington, D.C., 1988. IEEE Computer Society.

[Shi81] David W. Shipman. The functional data model and the data language daplex. *ACM Transactions on Database Systems*, 6(1):140–171, 1981.

[SHP88] M. Stonebraker, E. Hanson, and S. Potamianos. The POSTGRES rule manager. *IEEE Transactions on Software Engineering*, 14(7), July 1988.

[SHR87] R. Schulman and B. Hayes-Roth. ExAct: A module for explaining actions. Technical Report KSL-87-8, Stanford University, Palo Alto, CA, 1987.

[SK77] E. H. Sibley and L. Kerschberg. Data architecture and data model considerations. In *Proceedings of the National Computer Conference*, Reston, VA, 1977. AFIPS Press.

[SK86a] K. Schwamb and E. Kasif. OB1KB: A knowledge-based system for army order of battle intelligence analysis. Technical report, The MITRE Corporation, McLean, VA, December 1986.

[SK86b] A. Shepherd and L. Kerschberg. Constraint management in expert database systems. In L. Kerschberg, editor, *Expert Database Systems, Proc. from the First Intl. Workshop*, pages 443–468. Benjamin/Cummings, 1986.

[SLR88] T. Sellis, C. Lin, and L. Raschid. Implementing large production systems in a DBMS environment: Concepts and algorithms. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, New York, 1988. ACM Press.

[SLR89] Timos Sellis, Chih-Chen Lin, and Louiqa Raschid. Data intensive production systems: The DIPS approach. *ACM SIGMOD Record*, 18(3):52–58, 1989.

[SLU89] L. Stein, H. Liebermann, and D. Ungar. A shared view of sharing: the treaty of Orlando. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*. Addison-Wesley Publ. Co., Inc., Reading, MA, 1989.

[Sou87] J. M. Souza. *Software Tools for Conceptual Schema Integration*. PhD thesis, School of Information Systems, University of East Anglia, England, 1987.

[SR86] M. Stonebraker and L. Rowe. The design of POSTGRES. *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, May 1986.

[Sri88] S. Sripada. A logical framework for temporal deductive databases. In *Proc. 14th Intl. Conf. on Very Large Data Bases*, pages 171–182, Los Angeles, 1988.

[SS77] J. Smith and D. Smith. Database abstraction: aggregation and generalization. *ACM Transactions on Database Systems*, 2(2):105–133, June 1977.

[SS89] T. Sheard and D. Stemple. Automatic verification of database transaction safety. *ACM Transactions on Database Systems*, 14:322–368, 1989.

[Str88] B. Stroustrup. What is object-oriented programming? *IEEE Sofware*, pages 10–20, May 1988.

[Syn86] A. Synder. Encapsulation and inheritance in object-oriented programming languages. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 38–45, 1986.

[SZ86] A. Skarra and S. Zdonik. The management of changing types in an object-oriented database. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 483–495, 1986.

[SZ87] A. H. Skarra and S. B. Zdonik. Type evolution in an object-oriented database. In Bruce Shriver and Peter Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 393–415. MIT Press, Cambridge, MA, 1987.

[SZR86] A. Skarra, S. Zdonik, and S. Reiss. An object server for an object-oriented database. In *Proc. 1986 Intl. Workshop on Object-Oriented Database Systems*. ACM/IEEE, 1986.

[T+87a] M. Templeton et al. Mermaid – a front-end to distributed heterogeneous databases. *Proceedings of the IEEE*, 75(5), May 1987.

[T+87b] M. Templeton et al. Pragmatics of access control in mermaid. *Quarterly Bulletin of the Computer Society of the IEEE Technical Committee on Data Engineering*, 10(3):33–38, September 1987.

[Tem88] M. Templeton. Research areas in heterogeneous distributed DBMS. *Newsletter of the Computer Society of the IEEE Technical Committee on Distributed Processing*, 10(2):29–34, November 1988.

[Ull82] J. D. Ullman. *Principles of Database Systems*. Computer Science Press, Inc., Rockville, MD, 1982.

[Ull88] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, Inc., Rockville, MD, 1988.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-base Systems, Volume II: The New Technologies*. Computer Science Press, Inc., Rockville, MD, 1989.

[Urb87] S. D. Urban. *Constraint Analysis for the Design of Semantic Database Update Operations*. PhD thesis, Univ. of Southwestern Louisiana, 1987.

[Via88] V. Vianu. Database survivability under dynamic constraints. *Acta Informatica*, 25:55–84, 1988.

[VK86] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *American Association for Artificial Intelligence*, pages 377–382, Philadelphia, 1986.

[Wan89] Yair Wand. A proposal for a formal model of objects. In W. Kim and Frederick H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*. ACM Press, Addison Wesley, Reading, MA, 1989.

[Weg87a] P. Wegner. Dimensions of object-based language design. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 168–182, 1987.

[Weg87b] Peter Wegner. The object-oriented classification paradigm. In Bruce Shriver and Peter Wegner, editors, *Research directions in object-oriented programming*, pages 479–560. MIT Press, Cambridge, MA, 1987.

[Wei83] W. E. Weihl. Data-dependent concurrency control and recovery. *Proc. of the Second Annual ACM Symposium on Principles of Distributed Computing*, 1983.

[Wei88] W. E. Weihl. Commutativity-based concurrency control for abstract data types. *Proc. of the IEEE 21st Annual Hawaii International Conference on System Sciences*, 1988.

[Wer89] Charles J. Wertz. *The Data Dictionary Concepts and Uses*. QED Information Sciences, MA, 1989.

[Wie86] D. Wiebe. A distributed repository for immutable persistent objects. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 453–465, 1986.

[WK89] John R. Weitzel and Larry Kerschberg. Developing knowledge-based systems: Reorganizing the system development life cycle. *Communications of the ACM*, 32:482–488, 1989.

[Wol89] A. Wolski. LINDA: A system for loosely integrated databases. In *Proc. 5h Intl. Conf. on Data Engineering*, pages 66–73, Washington, D.C., 1989. IEEE Computer Society.

[WY88] T. Watanabe and A. Yonezawa. Reflections in an object-oriented language. *Proc. 3rd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 306–315, 1988.

[Y+85] C. T. Yu et al. Query processing in a fragmented relational distributed system: Mermaid. *IEEE Transactions on Software Engineering*, SE-11(8):795–810, August 1985.

[YA89] Jong P. Yoon and Anthony Arroyo. An object-oriented approach to rule-based constraint management. In *Annual Conference of the International Association of Knowledge Engineers*, College Park, MD, 1989.

[Yoo89] Jong P. Yoon. Techniques for data and rule validation in knowledge based systems. In *IEEE COMPASS*, pages 62–70, NIST MD, 1989.

[YT87] Stephen S. Yau and Jeffrey J. Tsai. Knowledge representation of software component interconnection information for large-scale software modifications. *IEEE Transactions on Software Engineering*, 13(3):355–361, March 1987.

[Z+86] C. Zaniolo et al. Object oriented database systems and knowledge systems. In L. Kerschberg, editor, *Expert Database Systems: Proceedings from the First International Workshop*, pages 49–65, Menlo Park, CA, 1986. The Benjamin/Cummings Publishing Company, Inc.

[ZW86] S. B. Zdonik and P. Wegner. Language and methodology for object-oriented database environments. *Proc. of the IEEE 19th Annual Hawaii International Conference on System Sciences*, 1986.

Appendix A Blackboard Architectures — Knowledge Sources and Control

The blackboard problem-solving paradigm has grown out of the AI field. It is typified by BB1 [HR85], a blackboard control architecture developed at Stanford.

BB1 (see Figure A1) provides a uniform blackboard architecture for knowledge-based systems that reason about their own actions. In a BB1 system, functionally independent knowledge sources cooperate by recording and modifying information in a global data structure called the blackboard. Task knowledge sources (also referred to as domain knowledge sources) solve problems on the corresponding task blackboard. Control knowledge sources construct control plans for the system's actions on the control blackboard. Learning knowledge sources modify information in the knowledge base. All knowledge sources operate simultaneously and, when triggered, compete for scheduling priority. BB1 also provides an explanation capability [SHR87] by which a system shows how its actions fit into its control plan [JHR86].

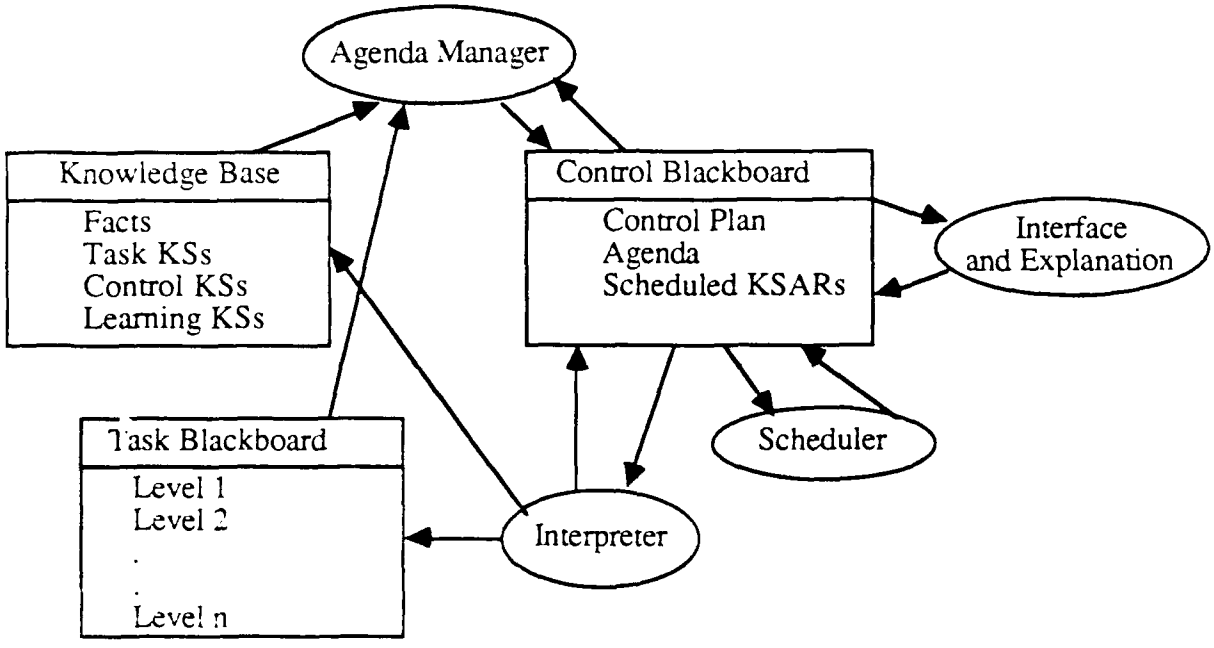


Figure A1. The BB1 Blackboard Control Architecture

Knowledge sources have a condition-action format. The condition describes situations in which the knowledge source can contribute to the problem-solving process. Ordinarily, it requires a particular configuration of solution elements on the blackboard. The action specifies the knowledge source's behavior. Ordinarily, it entails the creation or modification of solution elements on the blackboard. Only knowledge sources whose conditions are satisfied can perform their actions [HR85].

Each change to the blackboard constitutes an event that, in the presence of specific other information on the blackboard, can trigger (satisfy the condition of) one or more knowledge sources. Each such triggering produces a unique knowledge source activation record (KSAR). A KSAR is similar to an item on a task agenda. It represents a unique triggering of a particular knowledge source by a particular blackboard event. When a KSAR is chosen by the scheduling

mechanism, its knowledge source's action executes in the context of its triggering information, typically producing new blackboard events [HR85].

The system's control plan is managed and formulated on the control blackboard. Each control decision is represented as a data structure with the attributes defined in Table A.1. These attributes are instantiated for decisions at each level of abstraction on the control blackboard. The control blackboard's levels of abstraction represent different categories of control decisions. Decisions at the Problem, Strategy, Focus, and Policy levels describe desirable actions, thc by determining which of the system's control heuristics operate during particular problem-solving time intervals. Decisions at the To-Do-Set level describe feasible actions, identifying all KSARs eligible for execution on each problem-solving cycle. Decisions at the Chosen-Action level describe actions scheduled for execution on each problem-solving cycle. Figure A.2 defines and gives examples of decisions at each level of the control blackboard.

Table A.1 Basic Attributes of Control Decisions.

Attribute	Definition
Name	Identifying level and number
Goal	Prescribed action (predicate or function of KSAR attributes)
Criterion	Expiration condition (predicate)
Weight	Goal importance (0-1)
Rationale	Reason for Goal
Creator	KSAR that created the decision
Type	Role in control plan
Status	Function in control plan
First-Cycle	First operative cycle
Last-Cycle	Last operative cycle

The BB1 problem-solving cycle comprises three steps: (a) the interpreter executes the action of one task, control, or learning knowledge source, which changes the contents of the corresponding blackboard; (b) the agenda-manager adds KSARs to the agenda; and (c) the scheduler rates each KSAR on the agenda against the current control plan and chooses one KSAR to execute its action. Unless it has been instructed to operate autonomously, the scheduler also invites the user to request an explanation, to override its chosen action, etc. Thus, on each cycle, a BB1 system knows what actions are feasible (those on the current agenda) and what actions are desirable (those described in the current control plan). It determines which of its feasible actions best matches its desirable actions -- and performs that one [JHR86]. It is this flexible problem-solving cycle that enables BB1 systems to dynamically plan, execute, explain, and learn about their own actions.

A detailed description of BB1 is beyond the scope of this discussion. However, readers may consult [GCHR87, HR85, HHR87, JHR86, Nii86, SHR87] for further information.

The blackboard framework is regarded by many AI researchers as the most general and flexible knowledge system architecture. It offers expert system programming techniques that are difficult to achieve in other frameworks. Among them are:

- Dynamic control. At each step in the formation of the solution, a decision can be made as to how best to make inferences related to that step.
- Focus of attention. There is no rigidity with respect to what part of the emerging solution should be attended to next; for example, whether attention should go to an element at a low level of abstraction or at a high conceptual level.
- Flexibility of programming the control. Knowledge about how control should be applied in various domains can be codified in control rules or in complex control regimes.
- Modularity. Because the architecture is inherently modular (knowledge sources, blackboard levels, control structures, etc.) the design, testing and maintenance of the system can be eased.

The blackboard framework has been shown to be particularly well-suited to the classes of problems possessing one or more of the following traits:

- The need to represent many specialized and distinct kinds of knowledge.
- The need to integrate disparate information.
- A natural domain hierarchy (or hierarchies).
- Having continuous data input (e.g., signal tracking).
- Having sparse knowledge/data.

The problem of improving query specifications to a system of heterogeneous databases shares many of these traits. There are many specialized centers of knowledge to be represented. For example, there is the need to represent knowledge of query decomposition techniques, individual database contents, individual database query formulation methods, individual database contents from a system-wide viewpoint (i.e., an active knowledge encyclopedia of the databases' terms and of their usage), user desires and preferences, and past performance of databases.

Regardless of the data model utilized, databases are often conceptualized hierarchically. Also, it is natural for humans to conceptualize search for knowledge in a hierarchical fashion, and thus, it is desirable to guide the user in his quest hierarchically. The KDM allows one to easily represent hierarchies. The more obvious and natural the hierarchical representation, the more attractive the blackboard approach becomes.

Inherent in the heterogeneous nature of the system considered is the need to integrate disparate information. The blackboard provides an organizational framework to hold the fundamentally different kinds of information required to improve query specifications. Although not a part of this research, this blackboard feature could be applied in other capacities such as conducting system-wide query optimization.

Furthermore, it is likely such a system would possess sparse knowledge/data. Thus, there could be cases in which uncertain knowledge or limited available data would make absolute determination of a solution impossible (i.e., queries in which entity and attribute ambiguities were not fully resolved). In these cases the incremental problem-solving nature of blackboard systems still allows progress to be made. It is these reasons that make the blackboard framework appealing to the problem of improving query specifications to a system of heterogeneous databases.

Problem
  Definition: Problem the system has decided to solve.
  Example:
    Goal = (KS Problem-Domain = situation-assessment)
    Criterion =
      ((complete solution at lowest domain Level)
      (all Requested-Tasks planned)
      (all Constraints met)
      (route efficient))

Strategy
  Definition: General sequential plan for solving the problem.
  Example:
    Goal = (Action-Level = successive domain Levels)
    Criterion = complete solution spanning all domain Levels

Focus
  Definition: Local (temporary) problem-solving goals.
  Example:
    Goal = (Action-Level = Outcome)
    Criterion = complete solution at Outcome Level

Policy
  Definition: Global (permanent) scheduling criteria.
  Example:
    Goal = (Triggering-Cycle = recent)

To-Do-Set
  Definition: Sets of pending KSARs.
  Example
    Goal = ((Triggered-List = (KSAR-008,KSAR-009))
      (Invocable-List = (KSAR-005,KSAR-006, KSAR-007)))

Chosen-Action
  Definition: KSARs chosen to execute.
  Example:
    Goal = KSAR-007

Figure A.2 Levels of Abstraction for the Control Blackboard (From [HR85]).

B An example of schema evolution in GemStone

To illustrate one problem posed by evolving schemata in object-oriented databases, we created an example schema in the GemStone object-oriented database system and will present the transcript of a session we had with that system.

We first asked GemStone to tell us about its data dictionaries by entering the `list dictionaries` command at the system prompt as shown in line 1, below. The system responded by saying that it had dictionaries named `Globals`, `UserGlobals`, and `Example` as shown in lines 2 through 5.

```
1  topaz 1> list dictionaries
2  Symbol List dictionary names:
3  Example
4  UserGlobals
5  Globals
```

Next, we asked GemStone to list those classes defined in the dictionary named `Example`, for that dictionary is where we had earlier created a simple schema with which to illustrate the problem at hand. Line 1, below, shows our request. The system's response in lines 2 and 3 show that the dictionary contains two class definitions: one for a class named `Student` and another for a class named `Person`.

```
1  topaz 1> list classesIn: Example
2    Student
3    Person
```

Wanting to learn about the class `Student`, we asked GemStone to show the definition of that class. Line 1, below, shows our request and lines 2 through 22 show the system's response. From the system's response we learned the following about class `Student`:

- it is a subclass of class `Person` (line 3).
- it has one instance variable named `Advisor` (line 4).
- values of its instance variable are constrained to be of type `String` (line 8).
- it has two methods: one for updating the value of its instance variable (lines 11 through 16) and one for simply accessing the value of its instance variable (lines 17 through 22).

```
1  topaz 1> fileout class: Student
2  run
3  Person subclass: 'Student'
4    instVarNames: #( 'Advisor')
```

```
 5    classVars: #()
 6    poolDictionaries: #[]
 7    inDictionary: Example
 8    constraints: #[#[#Advisor, String]]
 9    isInvariant: false

10  %
11  category: 'Updating'
12  method: Student
13  Advisor: newValue

14     "Modify the value of the instance variable 'Advisor'."
15     Advisor:= newValue
16  %
17  category: 'Accessing'
18  method: Student
19  Advisor

20     "Return the value of the instance variable 'Advisor'."
21     ^Advisor
22  %
```

Wanting to learn more about class `Student`, we next asked GemStone about the class `Person` from which it inherits instance variables and methods. Line 1, below, shows our request and lines 2 through 22 show the system's response. From the system's response we learned the following about class `Person`:

- it is a subclass of GemStone's root class, `Object` (line 3).
- it has one instance variable named `Name` (line 4).
- values of its instance variable are constrained to be of type `String` (line 8).
- it has two methods: one for updating the value of its instance variable (lines 11 through 16) and one for simply accessing the value of its instance variable (lines 17 through 22).

```
1  topaz 1> fileout class: Person
2  run
3  Object subclass: 'Person'
4    instVarNames: #( 'Name')
5    classVars: #()
6    poolDictionaries: #[]
7    inDictionary: Example
8    constraints: #[#[#Name, String]]
```

```
 9    isInvariant: false

10  %
11  category: 'Updating'
12  method: Person
13  Name: newValue

14     "Modify the value of the instance variable 'Name'."
15     Name:= newValue
16  %
17  category: 'Accessing'
18  method: Person
19  Name

20     "Return the value of the instance variable 'Name'."
21     ^Name
22  %
```

From the metadata contained in dictionary **Example**, we concluded that class **Student** has two instance variables: **Advisor** which is defined as part of class **Student**, and **Name** which it inherits from class **Person**. Along with these two instance variables, class **Student** should have four methods: two for updating the values of the instance variables, and two for simply accessing their values.

To test this conclusion, we attempted to create an instance of class **Student** and assign values to its **Name** and **Advisor** instance variables. Lines 1 through 6, below, show our attempt to do this, and lines 7 and 8 show the error message generated by the system in response to that attempt.

```
1  topaz 1> run
2  | s |
3  s := Student new.
4  s Name: 'Chris Bosch'.
5  s Advisor: 'Dr. Kerschberg'.
6  %
7  [903, 10] No method was found for the selector #Name: when
8     sent to aStudent with arguments contained in anArray.
```

Contrary to what we had inferred from the metadata contained in dictionary **Example**, class **Student** does not have a method for updating the **Name** instance variable. Further investigation would show that class **Student** did not inherit any instance variables or methods from class **Person** as defined in the data dictionary even though it is defined to be a subclass of class **Person**.

The reason that this situation exists is due to the fact that the definition of class **Person**

has evolved over time. When we first defined class **Person**, it had two instance variables, **FirstName** and **LastName**, along with associated methods for updating and accessing those instance variables. Then, we defined class **Student** to be a subclass of class **Person**. Finally, we redefined class **Person** to have just one instance variable, **Name**, and associated methods.

Given this complete knowledge of the metadata's derivation history, we can conclude that class **Student** has three instance variables: **Advisor** which was defined as part of class **Student** as well as **FirstName** and **LastName** which it inherited from class **Person** as first defined. Along with these three instance variables, class **Student** should have six methods: three for updating the instance variables and three for simply accessing their values.

To test this conclusion, we attempted to create an instance of class **Student** and assign values to its **FirstName**, **LastName**, and **Advisor** instance variables. Lines 1 through 7, below, show our attempt, and lines 8 through 11 show the system's response indicating that our attempt was successful.

```
 1  topaz 1> run
 2  | s |
 3  s := Student new.
 4  s FirstName: 'Chris'.
 5  s LastName: 'Bosch'.
 6  s Advisor: 'Dr. Kerschberg'.
 7  %
 8  a Student
 9    FirstName       Chris
10    LastName        Bosch
11    Advisor         Dr. Kerschberg
```